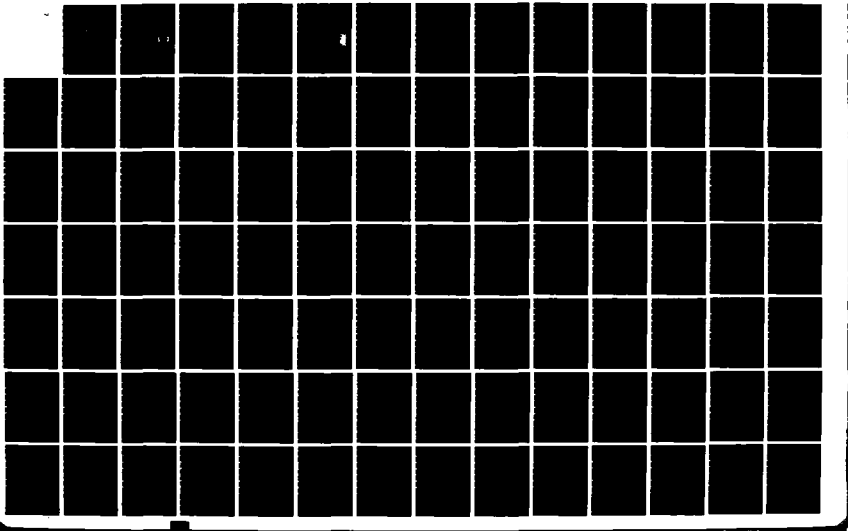
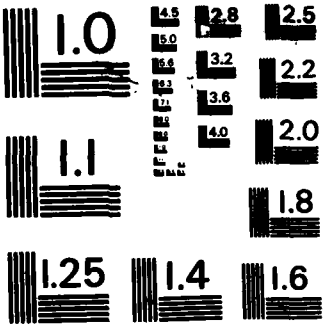
AD-A142 410 PERFORMANCE MEASUREMENT IN A DISTRIBUTED PROCESSING ENVIRONMENT(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH W E AVEN 1984 AFIT/CI/NR-84-10D 1/2

UNCLASSIFIED F/G 12/1 NL





UNCLASS

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

1

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER<br>AFIT/CI/NR 84-10D | 2. GOVT ACCESSION NO.<br>AD-A142 410 | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle)<br>Performance Measurement in a Distributed Processing Environment | | 5. TYPE OF REPORT & PERIOD COVERED<br>THESIS/DISSERTATION |
| | | 6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s)<br>William Eugene Ayen | | 8. CONTRACT OR GRANT NUMBER(s) |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS<br>AFIT STUDENT AT: The Ohio State University | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS |
| 11. CONTROLLING OFFICE NAME AND ADDRESS<br>/NR<br>B OH 45433 | | 12. REPORT DATE<br>1984 |
| | | 13. NUMBER OF PAGES<br>106 |
| ONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) | | 15. SECURITY CLASS. (of this report)<br>UNCLASS |
| | | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

STRIBUTION STATEMENT (of this Report)

ROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DTIC ELECTE JUN 26 1984 S D

B

STRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

APPROVED FOR PUBLIC RELEASE: IAW AFR 190-1

LYNN E. WOLAVER 15 May 84
Dean for Research and
Professional Development
AFIT, Wright-Patterson AFB OH

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

ATTACHED

AD-A142 410

DTIC FILE COPY

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE UNCLASS

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

84 05 21 136

# CHAPTER 1

# INTRODUCTION

## 1.1 ABSTRACT

A large class of distributed computing environments is emerging. These systems have a common set of distinguishing characteristics which separate them from single processor systems and many other distributed environments, and which must be considered when a measurement facility is designed for the new environments. The fundamental measurement questions of what to measure and how to measure are both impacted by the distribution, and these impacts occur in a number of ways and at various locations in the distributed systems. This dissertation describes the characteristics of a particular class of distributed computing environments and establishes the impact that the characteristics have on the design of a measurement facility for that class. The design of the measurement facility for a specific system in the class is presented.

AFIT/CI/NR 84-10D

AFIT RESEARCH ASSESSMENT

The purpose of this questionnaire is to ascertain the value and/or contribution of research accomplished by students or faculty of the Air Force Institute of Technology (ATC). It would be greatly appreciated if you would complete the following questionnaire and return it to:

AFIT/NR
Wright-Patterson AFB OH 45433

RESEARCH TITLE: Performance Measurement in a Distributed Processing Environment

AUTHOR: William Eugene Ayen

RESEARCH ASSESSMENT QUESTIONS:

1. Did this research contribute to a current Air Force project?

( ) a. YES ( ) b. NO

2. Do you believe this research topic is significant enough that it would have been researched (or contracted) by your organization or another agency if AFIT had not?

( ) a. YES ( ) b. NO

3. The benefits of AFIT research can often be expressed by the equivalent value that your agency achieved/received by virtue of AFIT performing the research. Can you estimate what this research would have cost if it had been accomplished under contract or if it had been done in-house in terms of manpower and/or dollars?

( ) a. MAN-YEARS _____ ( ) b. $_____

4. Often it is not possible to attach equivalent dollar values to research, although the results of the research may, in fact, be important. Whether or not you were able to establish an equivalent value for this research (3. above), what is your estimate of its significance?

( ) a. HIGHLY SIGNIFICANT ( ) b. SIGNIFICANT ( ) c. SLIGHTLY SIGNIFICANT ( ) d. OF NO SIGNIFICANCE

5. AFIT welcomes any further comments you may have on the above questions, or any additional details concerning the current application, future potential, or other value of this research. Please use the bottom part of this questionnaire for your statement(s).

NAME GRADE POSITION

ORGANIZATION LOCATION

STATEMENT(s):

FOLD DOWN ON OUTSIDE - SEAL WITH TAPE

AFIT/NR
WRIGHT-PATTERSON AFB OH 45433

OFFICIAL BUSINESS
PENALTY FOR PRIVATE USE, $300

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

**BUSINESS REPLY MAIL**
FIRST CLASS PERMIT NO. 73236 WASHINGTON D.C.

POSTAGE WILL BE PAID BY ADDRESSEE

**AFIT/ DAA**
**Wright-Patterson AFB OH 45433**

FOLD IN

# Performance Measurement in a Distributed Processing Environment

DISSERTATION

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

William Eugene Ayen, B.S., M.S.

* * * * *

The Ohio State University

1984

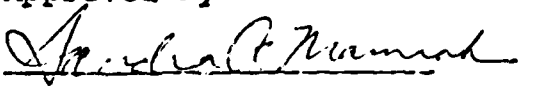
Reading Committee:

Dr. Sandra A. Mamrak

Dr. Bruce W. Weide

Dr. Dennis W. Leinbaugh

Approved By

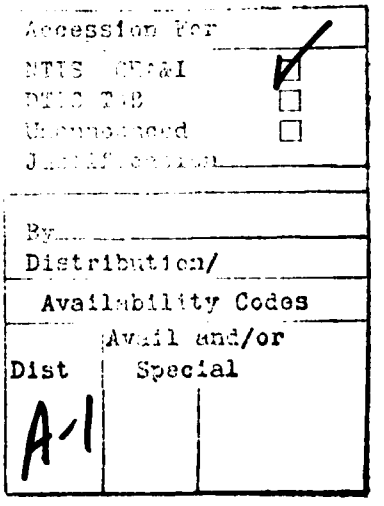


Advisor

Department of Computer and Information Science

## DEDICATION

I dedicate this dissertation to my family, Darcy, Mark, Kirsten, and Scott, for their never ending and unselfish support.

ACKNOWLEDGEMENTS

Without the support of my family and so many friends this work would not have been possible. My family has always been completely supportive of my efforts, understanding when they had no husband or father, and there when I needed them. My parents, Marion and Orville, gave me a tremendous start on life and have always been a source of inspiration. Many have provided special assistance - Dr. Ralph Curtis who started me on this path, Colonel Edmund Milauckas who helped me in my career, and Colonels John Wittry and Joseph Monroe who made this possible. A special thanks to Sandy Mamrak for the tremendous support and assistance she has given me over the last five years. Without her dedication and support I could not have succeeded. I also want to acknowledge the fine assistance provided by Bruce Weide and Dennis Leinbaugh in this effort. Most importantly, I want to thank God for always being present and giving me the strength to continue on.

## VITA

March 21, 1945 ..... Born - Monroe, Wisconsin

1967 ..... B.S., University of Wisconsin - Platteville, Platteville, Wisconsin

1968-1971 ..... Meteorologist, United States Air Force

1971-1972 ..... Graduate student, University of Missouri-Rolla, Rolla, Missouri

1972 ..... M.S., University of Missouri - Rolla

1973-1977 ..... Computer Performance Analyst, United States Air Force

1977-1979 ..... Instructor, U. S. Air Force Academy

1979-1981 ..... Graduate student, The Ohio State University, Columbus, Ohio

1982-1984 ..... Assistant Professor, United States Air Force Academy

## PUBLICATIONS

"Performance Measurement and Exception Handling in Desperanto's Distributed Environment.", Proceedings 3rd International Conference on Distributed Computer Systems, October 1982, pp. 847-853. With S. A. Mamrak, F. Gherfal, and D. Leinbaugh.

"A New Environment for Teaching Introductory Computer Science.", Proceedings 14th SIGCSE Technical Symposium on Computer Science Education, February 1983, pp. 258-264. With Sam Grier.

## FIELDS OF STUDY

Major Field: Computer performance evaluation
Minor Field: Computer networks

TABLE OF CONTENTS

LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION

## 1.1 ABSTRACT

A large class of distributed computing environments is emerging. These systems have a common set of distinguishing characteristics which separate them from single processor systems and many other distributed environments, and which must be considered when a measurement facility is designed for the new environments. The fundamental measurement questions of what to measure and how to measure are both impacted by the distribution, and these impacts occur in a number of ways and at various locations in the distributed systems. This dissertation describes the characteristics of a particular class of distributed computing environments and establishes the impact that the characteristics have on the design of a measurement facility for that class. The design of the measurement facility for a specific system in the class is presented.

## 1.2 MOTIVATION FOR RESEARCH

Computing environments formed by linking together existing computer sites by means of a communication subnetwork are currently an important class of computing environments. In these environments resources existing at an individual site can be made available to users at other sites. However, the level of resource sharing possible has been very limited [FOR78]. The computers involved are frequently of heterogeneous architectures and under managerial control of multiple organizations. These existing systems provide a wide range of existing software which is readily available only to users on the local systems. The communication subnetwork hardware is readily available and can be implemented with little difficulty. The software required to facilitate the sharing of resources across existing local systems, however, is much more difficult to design and implement. The software has been both difficult to write and hard to use and thus is not readily available. The difficulty arises because the software must interface existing operating system architectures that were initially developed under many different assumptions and in diverse environments.

Presently work is in progress at The Ohio State University to design a software support system for such distributed computing environments [MAM82a, MAM82b, MAM82c]. That design effort is focused on several issues of concern to designers. Among the issues being addressed are the design of the virtual interface for process intercommunication, naming, exception handling, and data conversion for resources included in the system, and the development of distributed applications that make direct use of the distributed environment [MAM83].

A very important issue that should be considered in the design of any system is the need to provide performance data. When this issue is included in the initial design of the system to be measured a number of benefits, such as availability of performance data throughout the system life cycle and more efficient implementation of the measurement capability, are possible. As a result, the design of a measurement facility and the issues pertaining to that design were investigated.

## 1.3 RESEARCH CONTRIBUTIONS AND ORGANIZATION OF DISSERTATION

The research for this dissertation resulted in several significant contributions related to the design and implementation of measurement facilities for systems in a class of distributed environments. The contributions are summarized in the following paragraphs which also provide an outline of the dissertation.

The class of distributed computing environments is defined in Chapter 2. This chapter establishes five distinguishing characteristics that separate systems in this class from other systems. These five characteristics,

-autonomous and heterogeneous computer sites

-lack of system wide coordination and control

-message passing used for interprocess communication

-disjoint memory address spaces

-distinct programmers-in-the-large and end users

define the class which provides a framework for the remainder of the research.

Given the definition of the environment, the next chapter establishes "what" to measure. The research identifies three measurement users - distributed systems analysts, programmers-in-the-large, and end-users. A major contribution of the research is the identification for the first time of the programmer-in-the-large as a measurement user. The research also establishes that the programmers-in-the-large and end-users have unique measurement requirements. These requirements, including the measurement parameters of interest, are described in Chapter 3.

The next chapter establishes "how" to accomplish the requirements identified in Chapter 3 given the environment defined in Chapter 2. A general model for measurement facilities is defined with four primary functions included in it: analysis, measurement, instrumentation, and control. A major contribution of this research is the identification of the interactions among these functions and how to design measurement facilities based on these interactions. The design approach is applied to systems under the constraints identified to yield for the first time the design of measurement facilities for this class of systems. The research establishes that existing measurement facilities cannot be used to satisfy the requirements outlined in Chapter 3.

The final major chapter, Chapter 5, specifies the measurement facility for an example system in the NAHCS class, Desperanto. The resulting design is simple. The facility utilizes servers to perform the measurement functions. During the definition of the servers a number of measurement primitives are identified. The identification of these primitives is a significant contribution since they can be used to implement measurement facilities in other systems in the class.

Further detailed specifications for the primitives and the measurement servers are given in the Appendices.

# CHAPTER 2

# A CLASS OF DISTRIBUTED ENVIRONMENTS

## 2.1 INTRODUCTION

Computer systems are comprised of processing, main storage, input/output, and auxiliary storage units. Early third generation computer systems contained a single processor, a large amount of core memory, and numerous peripheral devices with a single operating system managing the overall operation of the system. Multiprogramming allowed for parallel operations with the overlap of CPU and input/output (I/O) operations.

The development of mini- and later micro-computer systems resulted in a great increase in the availability of processors. Technology advances and increased processing requirements resulted in the development of computer systems with multiple processors sharing logical and sometimes physical resources. These systems have been labeled "distributed" meaning that not all of the processing functions are performed by a single processing element [BOO81]. Many different architectures have resulted [WEI80] as well as many

different definitions of what constitutes a distributed system [ENS78, TRI78].

A wide range of distributed system architectures can be defined. The specific architecture chosen for a system is greatly influenced by the environment in which the system is developed. This chapter introduces one class of distributed environments, those consisting of existing, autonomous and heterogeneous computer sites connected by a communication subnetwork. This Network of Autonomous and Heterogeneous Computer Sites (NAHCS) class of distributed processing systems provides a large number of potential benefits that have not existed in previous distributed systems [FOR78, MAM82b].

This chapter defines the distributed environment that contains systems in the NAHCS class. The class is distinguished by five specific characteristics that make it a unique class of systems:

a. existing sites are autonomous and heterogeneous

b. system wide coordination and control does not exist

c. interprocess communication is via message passing

d. memory address spaces are disjoint

e. programmers-in-the-large and end-users are distinct.

Each of these characteristics is discussed and example systems not contained in the class are specified.

## 2.2 DISTINGUISHING CHARACTERISTICS OF THE NAHCS CLASS

A large number of computing environments formed by linking together existing computer systems with a communication subnetwork exists. The difficulties encountered when interfacing existing operating system architectures designed under the normal assumptions of shared memory and global system state information has restricted the capabilities and structure of the existing interconnected systems. The software required to accomplish the interfacing is extremely difficult to write and not well understood [MAM83]. The distributed environments described below would allow for the sharing of existing resources while also facilitating the development of new, distributed services.

The design of the distributed systems in this class is based on the concept of guest layering. The motivation for the use of guest layering, the building of a uniform interface to existing local architectures, is that this approach provides for incorporation of existing resources into the distributed system without a need to modify these resources [MAM83]. Standard interfaces, defined for each system architecture, eliminate the need to define interfaces for each combination of architectures and eliminate the need to modify the existing system software.

The guest layer approach allows the users to selectively request use of the shared resources in the distributed system. The command language and the access mechanisms for the user are the same regardless of the location of the resource. Other approaches require the user to know the access mechanisms for both the network and the operating system of the host system where the resource resides.

The resources on the various systems do not have to be modified in order to be compatible with each other. Thus the software can be programmed to take advantage of local operating system services and achieve performance benefits. Current application and system software can be used without modification including the interfaces to the resources. The performance of the strictly local resources is not degraded since they are in no way changed by the guest layering.

The following sections define the NAHCS class by explaining the five distinguishing characteristics that make this a unique class. These characteristics distinguish systems in the class from those not in the class and impact on the design of the measurement facility for the class. Many systems have some of these characteristics, but no operational system meets all of the criteria for membership in the class.

### 2.2.1 Existing Sites

The NAHCS class contains systems formed by linking together existing single computer sites by means of a communication subnetwork. The computers at the individual sites are managed and operated by personnel at that site, not by a single managing entity for the entire system. Besides being autonomous the computers also can be heterogeneous - of different architectures and/or operating systems. Each computer site has a large base of existing software that has been developed to satisfy the needs of the local users. Each site's local users are unaffected by participation in the distributed environment unless they chose to participate in it.

The systems not in the class are distinguished by being under some degree of common managerial and operational control or containing homogeneous computer systems. Single processor systems are excluded by being owned and controlled by one organization and having a single processor. Multiprocessor systems, systems containing more than one processing element, are owned and managed by a single organization and can be dedicated to the accomplishment of a single task.

Excluded systems such as Cm* [SWA77], C.mmp [WUL72], and TI ASC [WAT72] are centrally controlled and have homogeneous processing components. Other distributed systems such as Tandem's Non-Stop systems [KAT78], Eden [LAZ81], RIG [LAN82], and RSEXEC [THO73] are excluded due to their common hardware base even though individual local systems may retain some degree of local autonomy. The Thoth system [CHE79] provides a portable real time operating system that is implemented on bare machines and that provides a homogeneous interface for all software resources on those machines. It is excluded since existing system services may not be executable on the system.

### 2.2.2 System Coordination/Control

The autonomous computer sites in the NAHSC class contain individual computer systems each with an operating system that controls the operation of only that individual system. No global operating system exists which controls the operation of the entire system or provides for naming, synchronization, resource management, or error handling.

Excluded systems have an operating system which controls at least in part the overall operation of the system. Single processor systems have a single operating system to control the entire computer system operation. The operating system for single processor systems maintains global system

state information that enables it to control the operation of the entire system.

Distributed systems excluded from the class have an operating system that controls the operation of the entire system and that maintains global state information. For example, the ILLIAC IV array processor has a control unit which controls the synchronous operation of the 64 processing elements [BAR68]. The GUARDIAN/EXPAND network operating system provides network control software for networks of Tandem/16 computers [BLA80]. The Cm* and C.mmp multiprocessor systems have had operating systems (Hydra [WUL81], StarOS [JON80], and Medusa [OUS80]) designed for them which schedule the various distributed system components.

### 2.2.3 Interprocess Communication

Two types of communication must be provided by local operating systems to support the guest layering approach. The first is the means for local processes to communicate. This is required to allow the standard interfaces defined to effect the guest layering to communicate with local processes. The second is the means to carry on remote communications with a process on another system connected to it by a communication link. This remote message passing capability must interface with the network services provided by the communication subnetwork.

A computer network has two principal components - the autonomous computer systems sometimes called hosts, and the communication subnetwork which connects the hosts together [TAN81]. The communication subnetwork consists of switching elements also called nodes, packet switches, or communication computers, and transmission lines, also called links, channels, or circuits. The switching elements control access by the hosts to the transmission lines to allow their orderly use in providing the reliable message passing capability of the network. All interactions between processes on separate computer systems must utilize this message passing capability.

The systems in this class use global network protocols provided by the communication subnetwork and remote communication capability provided by the existing local operating system to effect interprocess communication by message passing. The use of message passing causes delays and inefficiencies not always present in systems not in the class. Since the existing local systems can be heterogeneous, naming and other conventions will differ, thus increasing the effort required to effect the message passing.

For single processor systems the services provided by the local operating systems are used for interprocess communication. The single set of naming and protocol conventions of the system are utilized to easily effect the communication. The single systems generally allow communication between processes using "mail box" services provided by the local operating system.

Distributed systems not in the class may utilize the local communication capability of the operating system to communicate between processes which are logically distributed but which may or may not reside on the same processing node. In other systems the communication is accomplished using shared memory. The use of a single set of conventions and a common address space greatly reduces the problems of interprocess communication both from the standpoint of more easily implementing it and from considerations of efficiency and timeliness. Multiprocessor systems such as Cm* use shared memory to communicate. The Medusa operating system [OUS80] facilitates the interprocess communication by the use of a pipe mechanism similar to that found on UNIX systems.

### 2.2.4 Memory Address Space

The memory address spaces of the local computer systems are completely disjoint from each other. All communication between the systems must be done via message passing. Each autonomous system maintains its own system state tables and can only share this information with other systems using the message passing capability. As a result, state tables containing current status information on the distributed system cannot be maintained. The delays in updating any such tables and in accessing the tables are large.

For single processor computer systems each process can access the same memory address space with access to parts of that space restricted due to the different levels of privilege given the individual users. The operating system has the highest level of privilege and controls access to the system state tables.

Distributed systems outside the class may have at least some part of the memory space shared between the different processors. For example, Cm* has a single address space [RAS78]. Direct in-memory communication and the existence of global state tables is feasible with these shared memory systems.

### 2.2.5 Programmers-in-the-Large and End-Users

The NAHCS class is distinguished by the fact that there may be no organizational or application relationships between the owners and the end-users of those resources. The resources shared by the distributed system are initially developed by the owner programmer for use on the local system. This process is termed programming-in-the-small after DeRemer[DER76]. A resource is a set (possibly empty) of data objects and a set of associated operations on these objects. Other components, such as descriptive text, may be associated with the resource also. A resource may provide its operations to other resources and/or require operations from other resources. The existing resources are incorporated into the distributed system by the programmer-in-the-large (PITL). The end-users of the resources developed by the PITL need not be related to the PITL in any way. To the end-user the resources of the distributed system can be invoked as if the resources were on the local system. The only requirement is that the end-user be logged into the distributed system.

For single systems the resources are generally created, maintained, and used by the same individual or group of individuals. These programmers-in-the-small create the modules to satisfy a specific application requirement. Those resources which are more widely used will be made part of the local operating system services and thus be available to all users of that local system.

Distributed systems not part of the NAHCS class are frequently also designed for primary use on specific tasks. Due to the limited scope of use of the resources, the individuals developing them will also be the users. The particular architectural characteristics of the system are exploited to solve the specific task at hand. A common approach is to separate the task into subtasks each of which can be accomplished in a separate processing element with the accomplishment of the task managed by the global operating system. Examples include the use of task forces in the StarOS and Medusa operating systems for Cm* [GEH82].

Many current computer networks differ from the systems in this class since access to the resources available at remote sites is more difficult than for a local site. The PITL develops the resource and may want to permit access to the resources by remote users the same as on a single system. However, the remote user must first access the distributed system, then must "log" into the system where the

resource exists as an authorized system user and then invoke that resource using the conventions of that system. Thus, in systems such as the ARPANET, the user must take considerable action to gain use of the resources at another site. The NSW [FOR78] provides a standard interface but the programmer is limited to accessing only specific software development tools which have been made part of the system.

## 2.4 SUMMARY

This chapter defined the NAHCS class of distributed systems as a unique class of distributed systems. The five characteristics which distinguish this class of distributed systems were discussed. These five characteristics for the NAHCS class and example systems excluded from the class are summarized in Figure 2.1.

The characteristics that the distributed system must have to be part of the NAHCS class provides a framework from which to design a measurement facility. Many systems have some of the characteristics but the one or more characteristic that it lacks can have significant impact on the design of a measurement facility for that system.

Given the definition of the environment, the next task is to identify the potential users of the measurement capability and determine "what" to measure. Chapter 3 identifies three types of users and their measurement requirements. The identification of a new measurement user, the PITL, and the unique measurement requirements for these users are established. Chapter 4 then describes "how" the requirements can be satisfied in this environment. Chapter 5 presents the design for an example system, Desperanto.

| CHARACTERISTIC | NAHCS CRITERIA | EXCLUDED SYSTEMS BY THESE CRITERIA |
|---|---|---|
| Existing Site | Autonomous<br>Heterogeneous | Tandem [KAT78]<br>Eden [LAZ81]<br>RIG [LAN82]<br>Cm* [SWA77]<br>Thoth [CHE79]<br>RSEXEC [THO77]<br>C.mmp [WUL72]<br>TI ASC [WAT72] |
| System Coordination/ Control | Local | ILLIAC [BAR68]<br>Tandem [BLA80]<br>Cm* [JON80]<br>C.mmp [WUL72] |
| Interprocess Communication | Message passing | Cm* [SWA77]<br>Medusa [OUS80] |
| Memory Address Space | Disjoint<br>No global state | Cm* [RAS78] |
| PITL<br><br>End-User | Resource owner<br>Non-owner | Cm* [GEH82]<br><br>NSW [FOR78] |

Figure 2.1 Comparison of System Characteristics

# CHAPTER 3

# MEASUREMENT FACILITY REQUIREMENTS

## 3.1 INTRODUCTION

Measurement is the process of obtaining actual performance data from the system itself. It is an important aspect in the overall management of computer systems. Measurement of single processor systems has been an ongoing activity since the earliest systems. The technical literature contains many case studies and theoretical results and several books have been published that address issues in this area [BOR79, FER78, SVO76, DRU73]. For systems with multiple processors there are fewer published results [JON80] and the measurement issues are not as well understood.

A measurement facility is the logical combination of hardware and software that facilitates the obtaining of measurement data from a particular system. The facility can be a single physical entity such as a mini-computer based monitor which collects the data and performs analysis on it or it can be made up of a number of components which are logically connected but physically separated [NUT75].

Measurement facilities have been closely associated with the accounting function for some systems [IBM77, ROS78]. The requirements for cost accounting are very dependent on the overall system design. For the NAHCS class of systems, the accounting function has yet to be defined, and hence was not included in this effort. When defined for a particular system it could use part of the measurement facility to accomplish the accounting function.

This chapter establishes the measurement requirements for systems in the NAHCS class of distributed systems defined in the previous chapter. The first step in establishing the requirements is the identification of "who" the measurement facility supports. This research identifies three categories for systems in the NAHCS class - one which has not been previously recognized. For each category, measurement requirements are identified. After establishing the "what" in this chapter, Chapters 4 and 5 describe "how" the requirements can be satisfied in the general and specific cases, respectively.

## 3.2 MEASUREMENT SERVICE SUPPORT

Conceptually a computer system can be viewed as a tool which supports individuals by providing services to assist them in accomplishing their assigned tasks. The total spectrum of activities performed by individuals using computer systems can be functionally divided into two parts - that of providing services for others to use and that of requiring the use of these provided services (Figure 3.1). An individual may perform both activities at different points in time depending on the specific task being accomplished.

Measurement support is required for both activities. The specific measurement service required is dependent not only on the activity but also on the task being accomplished. This section briefly describes both the providing and requiring activities. It then identifies the specific functions which have measurement support requirements relative to the NAHCS class of systems. The specific measurement requirements for each of the functions are established in Section 3.3.

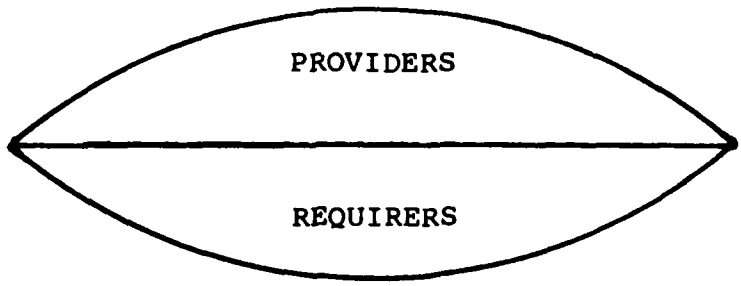



Figure 3.1 Division of Computer Usage by Activity

### 3.2.1 Service Providers

The providing services activity requires measurement support to assist in optimizing the performance of the service. For single computer systems all of the measurement support is often provided by the system analysts. They provide measurement support for all providers whether the service they provide is a system service such as the operating system or an application package like a data base management system (DBMS). It is convenient to divide the providing activity into two groups, the system providers and the applications providers (Figure 3.2).

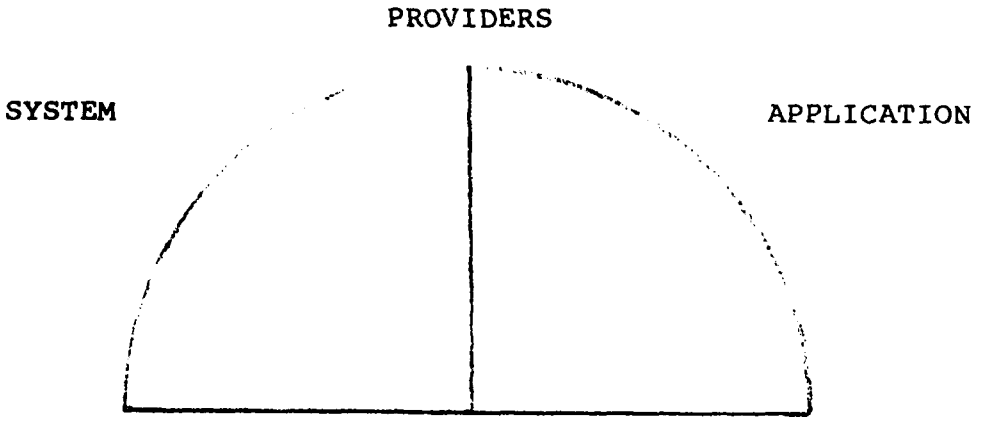



Figure 3.2 Functional Division of Service Providing Activity

The system providers provide services that facilitate the use of the computer system by all users. These services include those of the hardware components, the operating system, and the low level communications protocols. System providers are typically system owners and system or network analysts. Their measurement requirements are satisfied normally by system services provided by the operating system or by external measurement tools such as hardware monitors. The same measurement services are often provided as a system service to application providers.

The application providers provide application specific services to those users requiring them. They use the services provided by both the system providers and by other application providers to accomplish their tasks. The services they provide include compilers, text editors, DBMS's, and statistical analysis or simulation software packages. Application providers typically use operating system measurement services to satisfy their measurement requirements.

Distributed systems in the NAHCS class are implemented as a guest layer on existing, autonomous host systems. Hence, all NAHCS systems may be classified as "application" with respect to a given host. The application providers for the distributed systems can be logically divided into two categories which parallel the system and application providers activity division.

The first category includes the design, development, and maintenance of the distributed software system as an application program on a single host system. These distributed system analysts (DSA) are similar to the system providers described earlier in that they use the available services to develop services used by all distributed users. Since the services are developed as an application on each host, the measurement support required is similar to that for normal application providers on non-distributed systems.

The second category is the activity of defining the services provided in the distributed software system using programming-in-the-large [DER76] techniques. The programmer-in-the-large (PITL) incorporates individual resources into the distributed software system by defining the necessary interfaces. These resources can be either existing resources previously developed and now part of that system or new resources developed for use in this distributed system. The PITL may own the individual resources or can specify that a number of resources, some which are not owned by the PITL and are located at a remote host, are related in some fashion. This "system of resources" is treated as a single resource by the distributed system.

This is an activity new and unique to the NAHCS environment and, hence, measurement needs in this case have not been previously explored. This activity is similar to that of the application provider's on single systems, except now system service support comes through the distributed software system and not directly from the host system. The system of resources may span several individual computer systems.

Figure 3.3 shows division of application and system services providers. The DSAs and PITLs function in the NAHCS distributed environment while the service providers and other application providers do not.

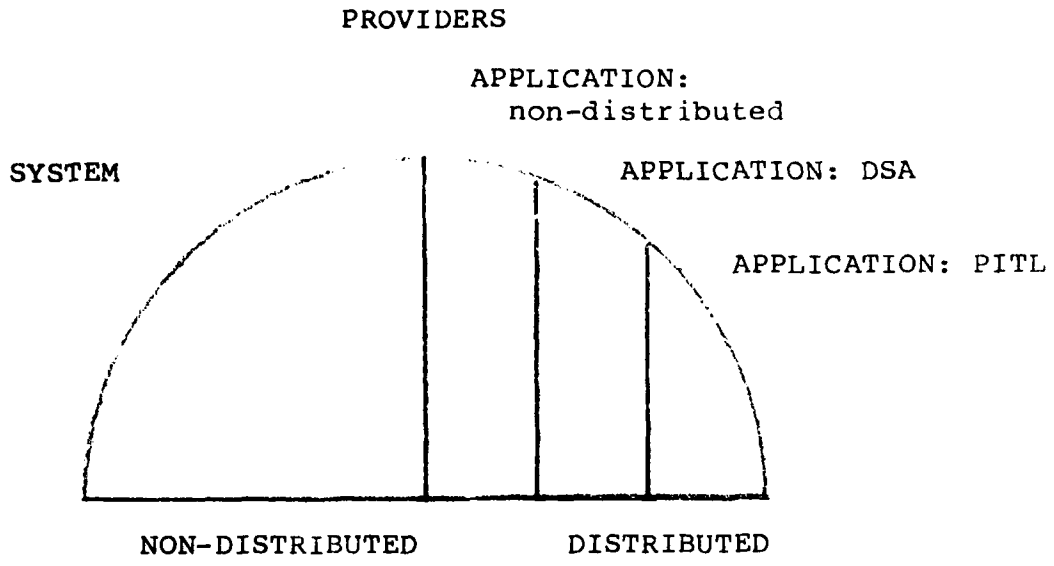



Figure 3.3 Further Division of Service Providing Activity

### 3.2.2 Service Requirers

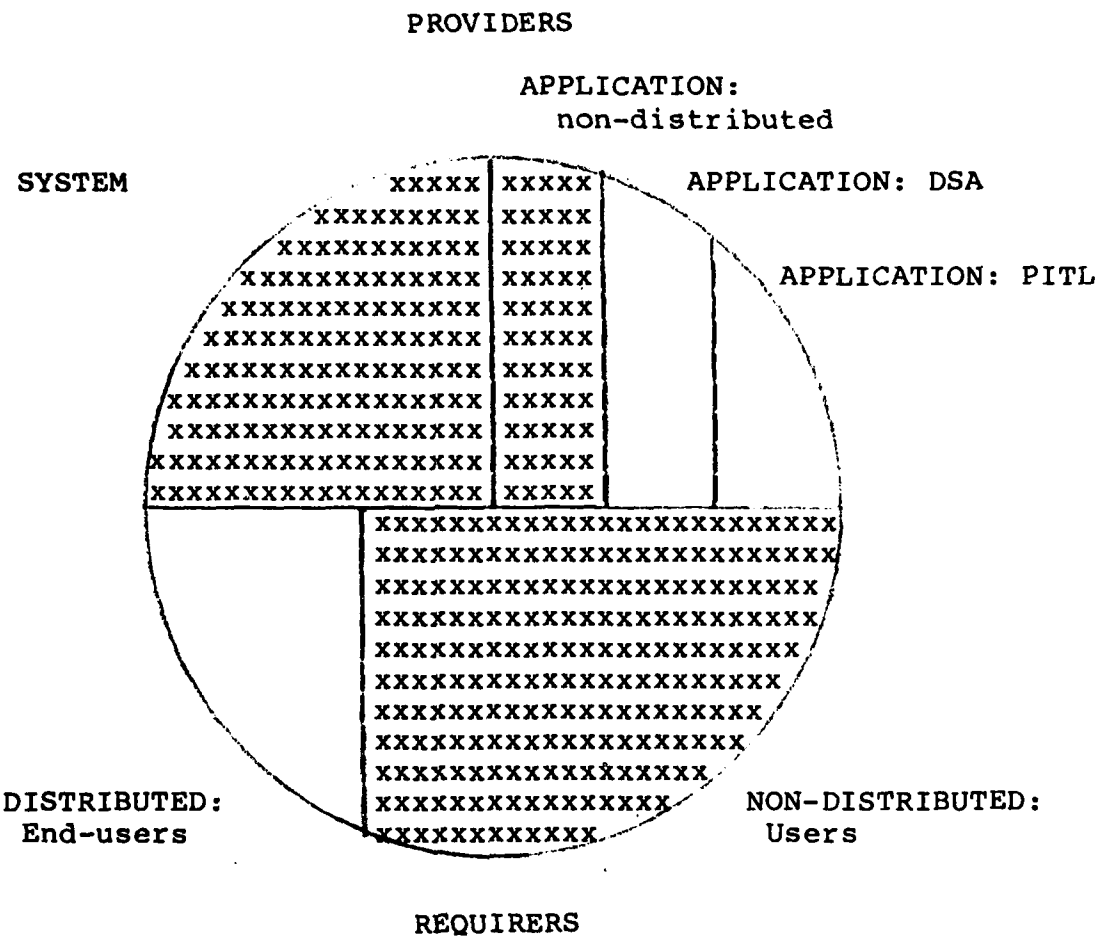
The second part of the activity decomposition is the requiring of services to accomplish specific tasks. Services utilized include text editors for program creation and compilers to translate programs into executable code which is executed using system services. Measurement services for most computer systems are provided by a system service, the accounting system, which provides data on the use of computer system resources.

For the NAHCS class the service requirers, while logged into the distributed software system, make requests for support from a resource that may be non-local. These end-users

require measurement support to assist them in making choices as to what alternative resources to use.

### 3.2.3 Summary

Figure 3.4 summarizes the measurement support requirements for the NAHCS class of systems. In particular, the functions which require support from the NAHCS measurement facility are shown. Measurement support not provided by this facility is provided by other existing measurement capabilities. The activities performed by individuals is divided into service providing and service requiring. Service providing is further broken into system and application providing. The system providers obtain their needed support from the system services of the host systems. Many of the application providers are not part of the distributed system and are excluded. The DSAs and PITLs do require support. Similarly for the service requirers, those activities not utilizing the distributed software support system do not require measurement support from the NAHCS measurement facility. However, the end-user of the distributed system does require measurement support. The next section discusses the specific requirements for each of individuals requiring support from the facility.




Legend: xxxx = measurement support already in place

Figure 3.4 Measurement Service Support for NAHCS Systems

## 3.3 MEASUREMENT REQUIREMENTS FOR NAHCS CLASS

The previous section identified three categories of individuals in the NAHCS class who have measurement requirements. This section describes the function of each of these categories and establishes their measurement requirements. Included in the description of the measurement requirements are the measurement parameters needed to satisfy the requirements. The measurement requirements are used in the next chapter to design the measurement facility for systems in the NAHCS class.

Measurement in the NAHCS class of systems is constrained by the environment in which the distributed systems are defined. These constraints affect both "what" can be measured and "how" it can be measured. The following gives a summary of the constraints organized by the five distinguishing characteristics for the class established in Chapter 2.

a. Existing sites: The autonomous local systems on which the distributed support software resides limit the information available to the measurement user. The end-user has access to resources located at remote sites only through the distributed support system, and hence will not have access to the same measurement information available if he/she were "logged" into the

remote site. Access restrictions can result from the requirement for special privileges which are not available to the end-user. Also, the PITL has limited access rights to resources she/he does not own. It may not be able to obtain all the measurement information desired, or even if the information can be obtained, the PITL cannot use the information to modify resources owned by others.

b. System coordination/control: Coordination between processes on separate hosts must be done using the network protocols. This implies that time delays of unknown length will occur if an event on one host results in a measurement activity at another host. Additionally, the clocks on the separate hosts are not necessarily synchronized thus causing errors in the measurement data if time stamps from two or more hosts need to be compared in some fashion. Measurement requirements are limited by these constraints. Analysis described in later sections has shown that this has no affect on the measurement requirements for systems in the NAHCS class.

c. Interprocess communication: All interprocess communication is done via message passing. This causes delays in the communication between processes. Also the additional communication required by the measure-

ment facility adds an amount of overhead to the system which can affect its performance.

In non-distributed systems the main memory is used for this communication. For many distributed systems there is ample communications capability to handle some additional traffic [SHO80]. Still, the side effects caused by the additional traffic must be considered in the analysis.

d. Memory address space: The lack of shared memory address space affects both the ability to communicate and to synchronize the measurements. These constraints are discussed in b. and c. above.

e. PITL and end-users: The requirement to support the PITL and end-users constrains the measurement requirements. The PITL, an application provider, and the end-user, a service requirer, have limited access rights to host computer systems in the distributed system. Consequently they are limited in the data accessible to them and in how they can use the data to improve the performance of the system.

The following three sections provide the measurement requirements for the measurement users in the NAHCS class of systems. Previous measurement efforts as well as the additional measurement areas for this class were considered in

the analysis. Given the constraints above and the NAHCS class characteristics, the following requirements are the total requirements for this set of measurement users.

### 3.3.1 Distributed System Analysts

The distributed system analysts design and maintain the distributed software support system. They create the distributed software support system as system programmers on the local systems. Some additional system privileges may be provided to these individuals. System services on the local system as well as the capabilities provided by the communication subnetwork will be the tools used by these programmers. The distributed system analyst performs functions similar to other system providers discussed in section 3.2.1.

The distributed system analyst will have available the system services of the host systems to satisfy many of his/her measurement requirements. These include accounting data, special operating system tools, and locally defined services [FER78]. Additionally, measurement capabilities provided as part of the communication subnetwork [COL71,SHO80] and related efforts in the area of distributed software development [SCH80, LAN82] are available. These measurement facilities are already in place and presented

extensively in the literature, and thus will not be discussed further.

### 3.3.2 PITL

The PITL requires measurement service to assist in managing the resources she/he owns. Management of a resource requires responding to changing requirements, both from external sources (end-users) and internal sources (actual execution needs).

External usage patterns may change leading to decisions to modify the resource to improve its performance. The usage patterns also provide the PITL with information needed to evaluate the impact of potential resource modifications to meet changing end-user requirements. Some modifications would be transparent to the end-user and thus have no impact, while others such as redefining the resource interface or removing the resource from the system would have major impact.

Internal activity, the characteristics of how the resource actually responds to a request for its service, may also vary with time. The PITL must analyze the performance of the resource to ensure that performance criteria are still being met. Increases in the response time on local or remote hosts or changes in the resources that provide services to this resource may affect the performance of the

owned resource.

Both the external and internal activity must be monitored to support decisions by the PITL. The measurement objectives that satisfy these requirements are described below.

PITL OBJECTIVE 1: Provide usage characteristics data for owned resources.

A single parameter provides the measurement data that satisfies this objective - "user count". For each end-user who requires a service provided by the resource, the number of requests for each service is measured. The length of time that the count is maintained is specified by the PITL.

PITL OBJECTIVE 2: Provide resource response data for owned resources.

The response of a resource to a request for a service it provides is measured by the "end-to-end response time". This parameter is the elapsed time between the receipt of a request for a particular service by the resource and the transmittal of the response to the requester. To satisfy the request the owned resource might request service from one or more other resources which in turn can make further requests for service.

The end-to-end response time parameter alone provides no information on how that time is spent. The response time can be divided into two components:

(1) the time that the owned resource is active serving the end-user request, the "active time", and

(2) the time that the owned resource is idle waiting for a response from the resource it requested service from, the "wait time".

The wait time includes end-to-end response time for requested services and the time involved in transmitting requests and responses from/to the resource, the "communication time". The actions which may take place and definitions for these times are as follows:

$t_1$ Resource A receives a request for the service it provides and begins servicing that request.

$t_2$ A requires the service provided by resource B located at a remote location relative to A. A requests the service of B and the request is transmitted to B.

$t_3$ B receives the request and begins servicing it.

$t_4$ B completes servicing the request and returns a response to A.

$t_5$ A receives the response and starts servicing the original request again.

$t_6$ A completes servicing the request and returns a response to the original requester.

Definition of parameters relative to resource A:

| | |
|---|---|
| End-to-end response time | $t_6 - t_1$ |
| Active time | $t_2 - t_1 + t_6 - t_5$ |
| Wait time | $t_5 - t_2$ |
| Communication time | $t_3 - t_2 + t_5 - t_4$ |
| Remote response time | $t_4 - t_3$ |

End-to-end response times, active time, and wait time parameters require measurement activity only relative to the owned resource. However, to determine how the wait time is spent requires measurement activity relative to non-owned resources. A lack of access rights and autonomy constraints can limit the amount of information which can be collected. As a result the granularity of the response time data is limited. This is a unique measurement characteristic of the NAHCS class of systems since typically the granularity of the measurement is only limited by the physical measurement facility, not by other constraints. The PITL can identify

that a bottleneck exists and based on available information can initiate discussions with other PITLs to look into the source of the problem. The PITL's decision making power is limited by this.

### 3.3.3 End-User

The end-users are the individuals who attempt to accomplish their specific tasks in an optimum manner using either existing resources available in the local or distributed environment or by creating modules in the local environment. The end-user requires measurement data to support decisions about which resources in the distributed environment to use.

The end-users may desire to obtain data on the performance of several resources, each of which could satisfy the user's needs. For example, multiple copies of a Pascal compiler may be available at different sites in the distributed system. To determine which compiler gives the best performance for the type of programs being developed by the user, measurements of performance statistics, such as CPU time, memory used, and response times, are necessary. When the best response time is identified, other factors such as cost and ease of use determined by non-measurement capabilities can be evaluated by the more sophisticated user to request the use of a particular compiler on a particular host. The same argument applies to other resources as well.

Typically cost accounting tends to be a system dependent function tied to usage patterns and to organizational philosophies. Historically this function has been separate from the measurement facility [DRU73, SVO76, ROS78]. Hence, cost accounting is not within the scope of this effort and will not be addressed further in the design of the measurement facility.

A single measurement objective satisfies the end-user requirements.

End-User Objective: Provide response time data for requested services.

The response time is the elapsed time from the request for service by the end-user until the service is completed and a response is transmitted to the end-user. This end-to-end response time for each service requested provides the end-user only with a measure of the performance of the service.

## 3.4 SUMMARY

This chapter established "what" to measure to satisfy the requirements for the NAHCS class of distributed environments. The requirements are based on the needs of the three identified measurement users - distributed system analysts, PITLs, and end-users. The distributed system analyst functions much like an application programmer and their requirements can be satisfied through existing measurement capabilities. The PITLs have never before been identified as a measurement user along with their requirements for managing resources. Finally, the end-users require response time measurements to satisfy their needs.

The specific measurement requirements are affected by the characteristics of the NAHCS class of distributed systems. The most significant affect is that the granularity of the measurement data which can be collected is limited by the constraints of the class.

The measurement requirements established in this chapter are utilized in the next step in the design process, the "how" of the measurement problem. The next chapter establishes the design approach and Chapter 5 specifies an example implementation.

# CHAPTER 4

# MEASUREMENT FACILITY DESIGN

## 4.1 INTRODUCTION

A methodology for the design of measurement facilities is an area which has received very little attention in the past. Most measurement facilities have been developed in an ad hoc fashion to meet the immediate needs for already existing systems. The resulting measurement facilities are inefficient and do not provide all the desired information.

Previous chapters have established the environment and the requirements for the measurement facility in the NAHCS class. This chapter introduces a measurement facility model and describes how it can be used during the design of measurement facilities. A design approach utilizing this model to design a measurement facility for the NAHCS class that satisfies the requirements from Chapter 3 is presented. The general design is used in the next chapter to design a facility for a specific example distributed software system.

## 4.2 MEASUREMENT FACILITY DESIGN APPROACH

The design of a system begins with identification of the functions the system is to perform. Once the functions are identified the system design process starts at the highest level and then proceeds in a step-wise fashion to lower and lower levels until the design is complete. This is similar to the design approach for any software system as well as for systems in general.

A measurement facility can logically be divided into four distinct functions - analysis, measurement, instrumentation, and control. This research for the first time identifies the interactions among these functions and defines a model describing these interactions. The model serves as the basis for the design approach for measurement facilities.

### 4.2.1 General Measurement Facility Model

The four main functions and interactions among the functions are represented by the model of the measurement facility given in Figure 4.1. This model presents the flow of measurement data from the system under study through the measurement facility and the flow of control in the measurement facility. The four functions present in a measurement facility are not necessarily uniquely identifiable as

separate physical components but are logically present. A brief description of each of the functions follows.

The control function is the interface between the user and the other components of the facility. It receives commands from the user either interactively via a display device or non-interactively from a batch type input. The commands may initiate or terminate a measurement session, provide options that control the collection of measurement data, or provide options for the analysis of measurement data. In an interactive environment, the control function asks the user for commands and provides intermediate feedback to the user on the status of the measurement session.

The analysis function provides the data reduction capability necessary to process the data collected by the measurement function and to provide the information needed to satisfy the user's measurement objectives in the form of measurement reports. The analysis is accomplished under the control of the user and may be physically done on the system under measurement or may be done on a separate system.

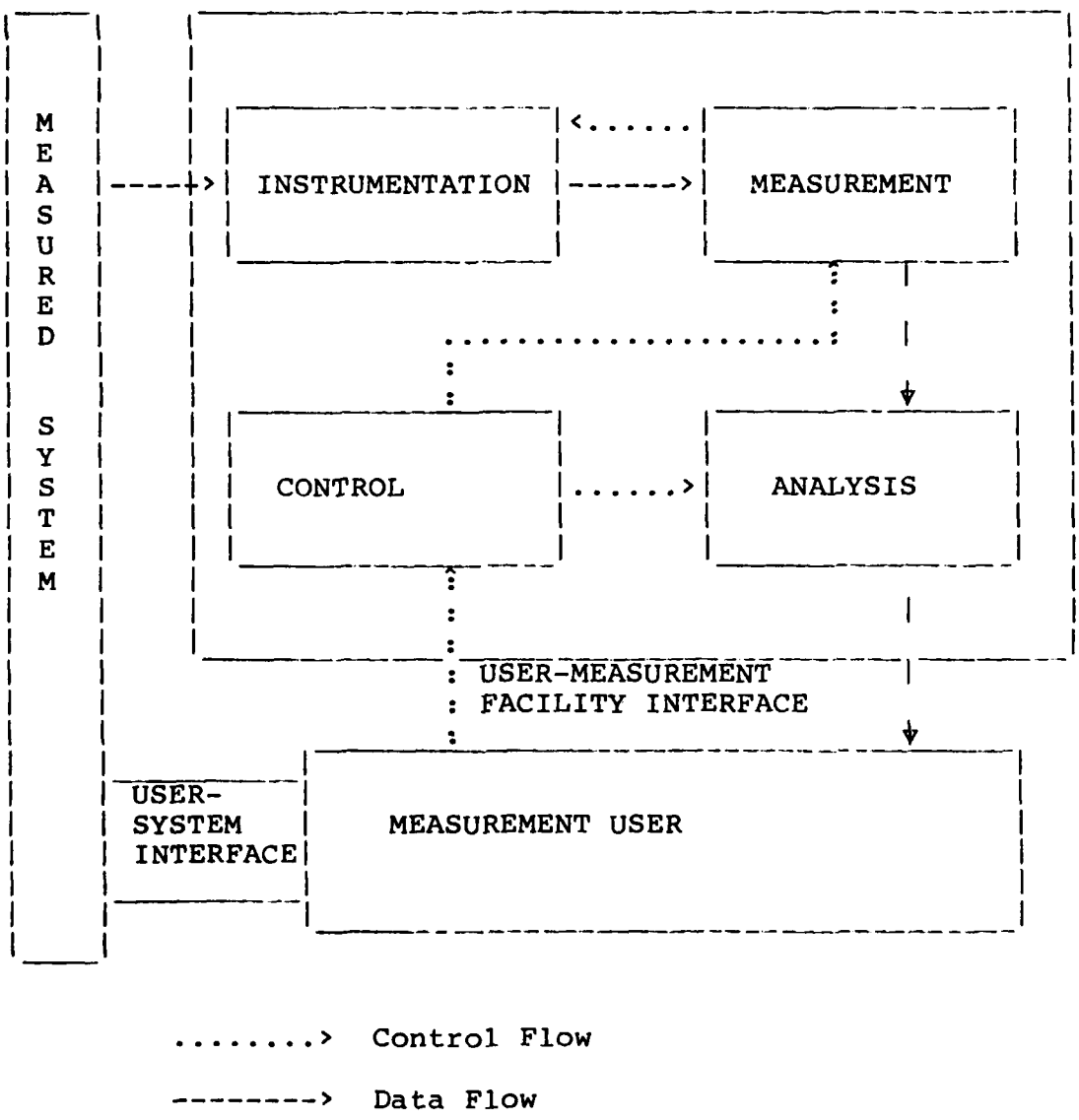



Figure 4.1 Measurement Facility Model

The instrumentation function provides the interface to the system under study. It detects the signals/events that represent the activities of interest and provides the measurement function with the appropriate data. This function may be implemented in hardware or software.

The measurement function is a key function in the facility. It receives data from the instrumentation function, determines the action to take with respect to the data based on commands from the control function, and provides the resulting data to the analysis function.

### 4.2.2 Use of Existing Measurement Facilities

Consistent with the philosophy of systems in the NAHCS class, the use of existing measurement resources were investigated. The following briefly surveys representative measurement facilities which currently exist. These facilities include examples from both distributed and non-distributed environments.

Several sophisticated measurement facilities based on the software monitor concept have been implemented for single processor systems [ROS78]. These facilities are system specific and cannot be transported to systems other than the ones they were designed for. Furthermore, measurements taken by one facility/monitor cannot necessarily be compared to those taken by another due to differences in definition of

terms and in architectural differences. IBM's Resource Management Facility(RMF) [IBM77], Univac's Software Instrumentation Package(SIP) [UNI78], and the Generalized Monitor Facility(GMF) [WAL80] for the Honeywell 6000 series computers are examples of these facilities. These facilities provide information primarily on the utilization of the system resources. CPU busy, CPU wait, channel busy, CPU/channel overlap, memory reference map, I/O data set access patterns, job statistics, and system trace data are examples of the type of data collected by these facilities.

Measurement facilities for distributed systems are relatively few compared to the work which has been done for single processor systems. The following examples illustrate approaches to the design of measurement facilities for distributed systems.

ARPANET: The ARPANET, a long haul network, included a Network Measurement Center(NMC) with the responsibility for defining and controlling the measurements needed to support the modeling efforts for the network and to determine the performance of the network [COL71]. The measurement facility component had extensive measurement capabilities located in the IMP's, the message switching computers located at the distributed sites. The measurement facilities accumulate data on the overall performance of the network or some

segment of the network, obtain data at an IMP relative to queue length and routing information, and obtain traces of the messages as they flow through the network. The emphasis of the measurements was on optimizing the flow of messages through the network.

Computer Network Monitoring System(CNMS): The CNMS was designed at the University of Waterloo for the purpose of monitoring the states and activities of each node in the network and of the communication links between the nodes [KOL77, BUC77, MOR75]. The measurement system was designed to be used for a local area network and included in its components a remotely controlled hybrid monitor at each node to gather data on that node. The operation of these monitors were under control of a mini-computer, called the controller. The user determined what information was desired and the controller then directed the individual monitors. The components of the CNMS were envisioned to be minicomputers external to the network connected together by communication links and connected to the network components by probes.

Ethernet: The performance of Ethernet, a local area network using a carrier sense multiple access with collision detection(CSMA/CD) protocol, was studied by Shoch [SHO80]. The purpose of the study was to determine the performance of the communication subnetwork, a high speed bus, testing the

CSMA/CD protocol under different load conditions. The measurement facility was implemented by dedicating one node on the network as a promiscuous node that observed all activity on the communications subnetwork and recorded data of interest. The performance study provided information on network delays, traffic distributions, traffic types, and collision and error rates.

Network Measurement Machine (NMM): The NMM was designed at the National Bureau of Standards for the purpose of measuring the performance of computer networks in an interactive environment in terms of the service they provide [ABR79]. The NMM was implemented on a PDP 11/20 with both regular and special purpose hardware. The special purpose hardware was used to connect the NMM to the network. The parameters of interest, the response time, turnaround time, and throughput, are used to measure the service provided to the user by the network, but not the internal operation of the host computer systems.

Later measurement work at NBS was done by Amer [AME82]. His local area network measurement center collects performance data on the traffic on the network such as network delays, traffic distributions, and types of traffic.

National Software Works(NSW) - Foreman: The NSW Foreman included a Performance Measurement Package(PMP) that provided basic resource utilization characteristics of programs [SCH80]. The system was designed for the TENEX/TOPS-20 operating system and provided utilization statistics for individual programs. System time, CPU time, system load, group load, and pager statistics were measured for each event selected for measurement.

Figure 4.2 provides a summary of these measurement facilities. The existing facilities are doing a good job at meeting the requirements they were designed to satisfy - system resource utilizations. However, the measurement objectives for the facilities differ from those of the NAHCS facilities. Consequently, they do not measure the right things and so are inadequate for our needs.

| FACILITY | OBJECTIVE | SYSTEM REQUIREMENTS | PARAMETER ORIENTATION |
|---|---|---|---|
| RMF<br>GMF<br>SIP | System tuning | Operating system<br>Privileged user | Job and resource utilization |
| ARPANET | Network modelling & evaluation | Instrumented IMP | Network traffic |
| CNMS | Node and link evaluation | External to network | Resource utilization |
| Ethernet | Network evaluation | Ethernet | Network utilization |
| PMP | Program evaluation | National Software Works | Program utilization |
| NMM | Network service | External to network | Response time<br>Throughput |
| NAHCS Class | Resource management<br>End-user support | None<br>None | Response time<br>Usage<br>Response time |

Figure 4.2 Comparison of Measurement Facilities

## 4.3 NAHCS MEASUREMENT FACILITY DESIGN

The general design of the measurement facility for the distributed system considers the two categories of individuals requiring distributed measurement support, PITL and end-users, their requirements, and the distributed environment that it is designed to function in. The general design presented in this section serves as a model from which specific facilities can be designed and implemented for systems in this class.

The approach to the design of measurement facilities in general was developed during the early part of this research. The methodology was successfully applied to the design of measurement facilities for three different distributed systems. The details of the methodology and the results from applying it have been documented in a Department of Computer and Information Science internal document. The validity of the approach and the benefits from using it were established in these efforts.

The approach utilizes the measurement facility model and the flow of data within the facility (Figure 4.1). Starting with the analysis function which satisfies the measurement requirements, the design sequence proceeds logically in the direction opposite to the flow of data in the facility.

### 4.3.1 Analysis

There are two design options for the analysis function for any system. The first approach is to collect all measurement records at specific physical locations in the system. The analysis can be performed at these locations or at another location. A second approach is to collect and analyze the data at the location where the measurement function resides, thus reducing the system overhead involved in moving the records to a different location.

The analysis function for distributed systems in the NAHCS class can be designed using either approach described above. For most measurements the analysis can be accomplished at the location where the measurement function resides. This assumes the requisite capability at that location to support implementation of the analysis function. If the capability does not exist at all sites or if the data collected at multiple locations must be combined to obtain a value for the required analysis parameter then the first approach is needed. This implies a need for consistent meaning for the data collected and may require calibration of clocks between the individual measurement facilities.

### 4.3.2 Measurement

The measurement function obtains, processes, and records the data from the instrumentation function and provides it to the analysis function under direction of the control function. Its design is influenced to a great extent by the structure of these other functions with which it interfaces. The measurement requirements of the PITLs and end-users are significantly different and are described in the next two subsections.

#### 4.3.2.1 PITL Measurement Design

The measurement requirements for the PITL are to provide usage and response time data for the owned resource. Data for those parameters which require no information other than on the resource activity can best be obtained at the resource itself. By placing the measurement function adjacent to the resource being measured all requests for service from a resource, all responses from the resource, and all requests from the resource can be detected by the measurement function.

To determine information on the components of the end-to-end response time, when the owned resource makes a request for service from another resource, measurement actions must be taken on other than the owned resource. To collect

this data, called trace data, a slightly more complicated design is required. The design of the trace facility is complicated not only by the fact that the trace cannot be done strictly locally, but also that a request for a given operation may travel not only to resources which are owned by the individual making the measurement request, but also directly or indirectly to other resources and to resources that are strictly local and have not been installed into the distributed environment.

Figure 4.3 illustrates that a resource may require services from resources owned by other PITLs. In this case, resource A requires service from resource B. Resource B in turn requires service from resource C to satisfy the request from A. Resource B has a different owner than A and C may have yet another owner. The owner of B has restricted the amount of information that is available to users of that resource. The restriction could be for privacy or security reasons or due to the unavailability of the data.

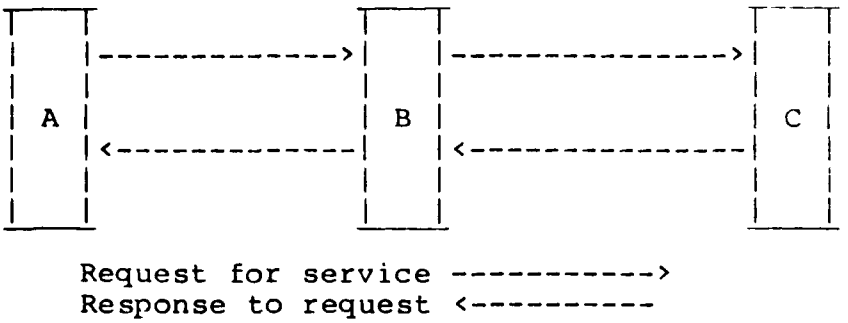



Figure 4.3 Example of Resource Interaction

The trace capability is designed as a partial one, guaranteeing end-to-end information, as well as measurement data on all distributed resources traversed for which the measurement requester has the proper access rights. The measurement function is distributed across nodes in the system.

The measurement function is activated by the PITL requesting measurement support from the distributed system. The options specified in the request determine the type, duration, and disposition of results for the measurement session.

#### 4.3.2.2 End-User Measurement Design

The measurement requirement for the end-user is to provide end-to-end response time for requested services to the user. Since the parameters are concerned with the interaction of the end-user with the distributed system, the design simply provides for placing the measurement function at the point where this interaction takes place. When a distributed user enters the distributed environment by "logging" into the distributed system a terminal server is created which henceforth acts as the interface between the user and the distributed system, including the measurement function for the end-user.

The measurement function is activated at the request of the end-user and will exist for only as long as the end-user desires. The end-user can specify measurement options such as length of measurement, parameters of interest, and display of results desired.

### 4.3.3 Instrumentation

The instrumentation function provides the interface of the measurement function to the measured system and hence its design is influenced by the structure of both. For distributed systems this function is usually distributed. Once the parameters have been identified and the measurement function designed the method of extraction of the data can

be determined. For the NAHCS class it is implemented as part of the distributed system software. The detection and recording of the measurement data should have the least impact on the system and yet provide the most complete set of data possible.

### 4.3.4 Control

The control function interfaces the user to the measurement facility and controls the collection and the subsequent analysis of the data. The control function design depends very heavily on the design of the measurement and analysis functions as well as the user interface to the facility.

The control function for the measurement facility in this class will use the framework provided for the distributed system design. It will be implemented as a resource on individual local sites in the system.

## 4.4 CONCLUSION

This chapter presented the general design for measurement facilities in the class of distributed environments introduced in Chapter 2. A general measurement facility model and design methodology for all systems was introduced. The design methodology was used to design this facility.

For these distributed systems all functions with the possible exception of the analysis function will be distributed. This is consistent with the overall design of the distributed system. However, even though the functions are distributed they act independently at each site with no central control function for the measurements facility.

The next chapter utilizes the general design to give a more specification of the measurement facility for a specific example system in this class. The example system, Desperanto, is described and the approach to implementing the facility for the Desperanto system is given.

# CHAPTER 5

# MEASUREMENT FACILITY IMPLEMENTATION

## 5.1 INTRODUCTION

The previous chapters established the requirements for and the general design of a measurement facility for the NAHCS class of distributed environments. This chapter describes the design for an example system in this class. This example system, Desperanto, is under development at The Ohio State University [MAM83]. The design is based on the general design from Chapter 4. Existing measurement facilities are not able to support the measurement facility requirements. The measurement functions are performed by servers which can be implemented using measurement primitives defined in this chapter. Detailed specifications for the servers are given in Appendix B.

## 5.2 DESPERANTO SOFTWARE SUPPORT SYSTEM

Desperanto, an Esperanto for Distributed Systems, is a comprehensive software system which has been designed to support the sharing of resources in the NAHCS class of systems described in Chapter 2 [MAM81, MAM82b]. The design effort has been ongoing at The Ohio State University for over three years and implementation is proceeding.

A model of the Desperanto system is provided in Figure 5.1. The sharable resources reside at various locations in the local area network. Each location/site in the network is connected to the communications subnetwork which provides reliable message passing capability. At each site Desperanto resides as a guest on the local operating system, running as one or more processes under constraints of that local operating system. Each site has a standard Desperanto monitor which provides run-time support including process management, communication, directory assistance, deadlock analysis, etc. in cooperation with the standard monitors at other network sites. Implementation of the standard monitor is system dependent and may require several processes to perform the various functions required of it. It is located at the Presentation Layer as defined by the ISO Network Architecture [DES80,ZIM80].

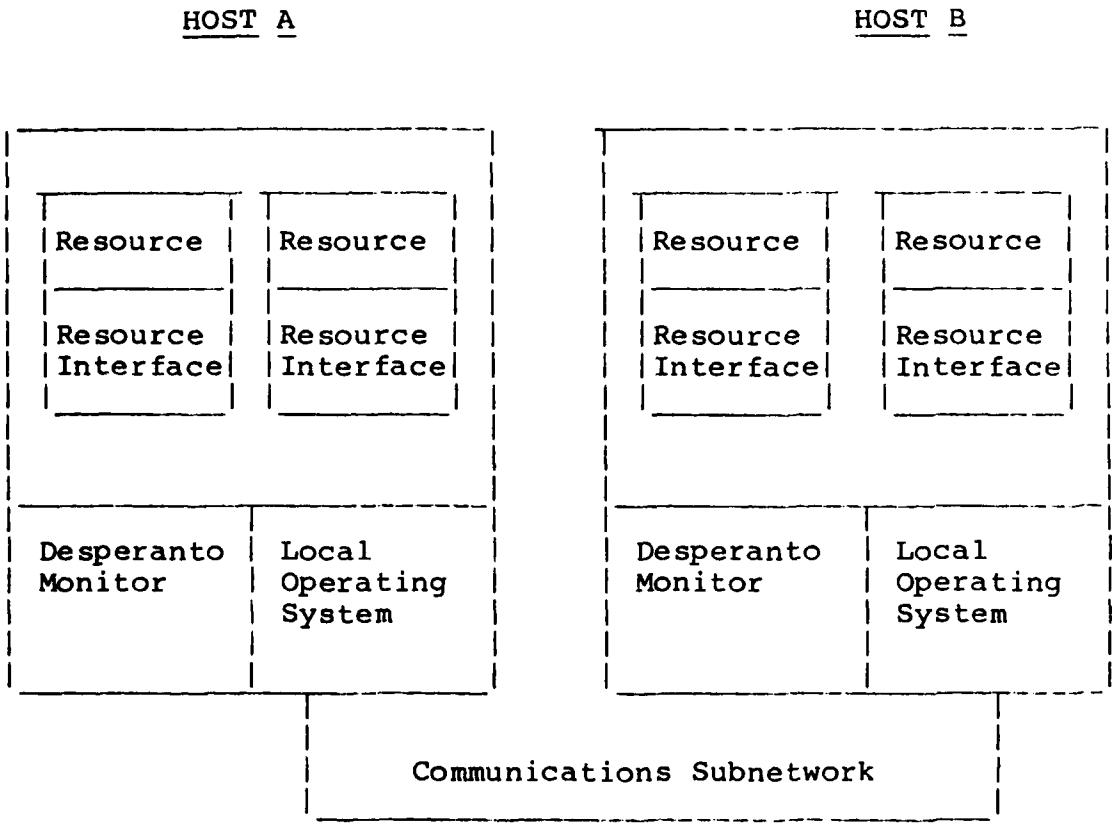



Figure 5.1 Desperanto System Model

Each resource has a unique logical server which creates a server process to perform the resource operations and which provides resource unique services. The resource-dependent information contained in the resource interface is provided by the programmer who is responsible for programming-in-the-large, the individual who specifies the resource to Desperanto.

An individual user of any computer system in the network will normally use the native system without support

Each resource has a unique logical server which creates a server process to perform the resource operations and which provides resource unique services. The resource-dependent information contained in the resource interface is provided by the programmer who is responsible for programming-in-the-large, the individual who specifies the resource to Desperanto.

An individual user of any computer system in the network will normally use the native system without support from, or in fact knowledge of, Desperanto. The user's application may require services from another system which can be provided by user defined protocols or other existing communications capabilities. If a user choses to take advantage of the facilities of Desperanto then he/she must "log" into the Desperanto system. Requests for support from resources known to Desperanto will be handled by the Desperanto monitor. A terminal server process provides a "friendly" interface to the Desperanto system for all users.

Desperanto provides a modular, extensible set of common services to application programmers and a coherent and consistent interface to end-users, all without change to existing local software.

### 5.2.1 Example Desperanto Interaction

To illustrate the operation of the Desperanto system an example session is described in this section. Figure 5.2 illustrates the interactions in the Desperanto system for this example. It is assumed that the standard servers at two sites, A and B, are already running. They are denoted by A1 and B1 respectively. Distributed addresses or ports for all the processes known to Desperanto have unique mailbox id's. At site A a server is currently running in support of a request from a user. In this example this server requests service from a resource which is located at site B. The server for the resource at site B may or may not be currently running or active. The interaction would be as follows:

1. Requester A3 notifies the standard server, A1, that it requires service from resource B3 at site B.
2. Standard server A1 creates a process to run A3's module interface and passes the mailbox/port id to this process, A2.
3. The requesting server acknowledges the module interface process when it is notified of it's existence and passes a request for support from resource B3 to process A2.

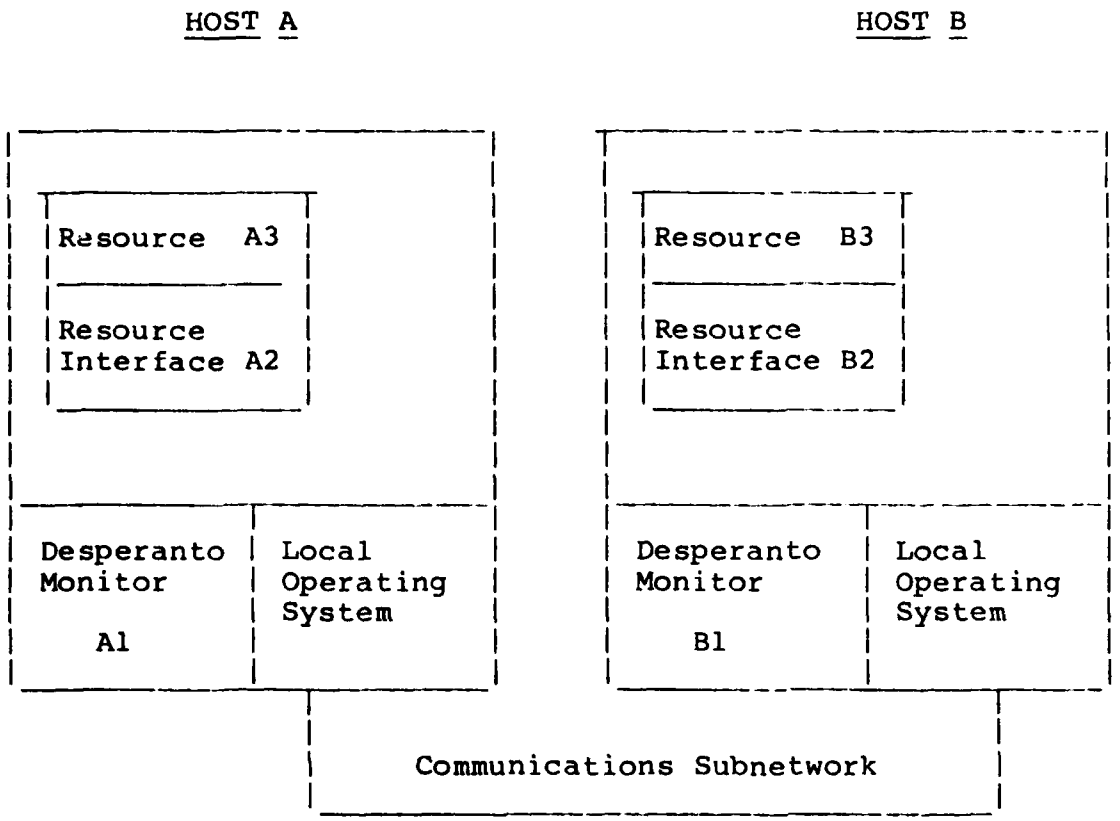



Figure 5.2 Desperanto Interaction Example

4. Module interface A2 notes the request and forwards it on to A1.
5. Standard server A1 notes the request and determines the location of the requested resource (B). The request is sent by A1 to B1 using the message passing capability of the subcommunications network.
6. Standard server B1 notes the request, determines the correct resource interface, creates a process(B2) to run the interface, if it is not already running, and passes the request to B2.
7. Tool interface B2 notes the request, determines the proper server process, and creates it (B3). The request is then passed to B3.
8. Provider server B3 acknowledges the connections, passes its port or mailbox id to B2.
9. The connections are noted and a message with needed connection information is forwarded from B2 to B1 to A1 to A2 to A3, in turn.
10. Requester A3 then sends the request to B3 using the connection information it has just received.

11. Provider B3 then processes the request during which time it may make one or more distributed requests for service from other resources known to Desperanto. When the request has been completed the results are returned to A3.

At this time the connection could be broken or it could be retained for further requests. The process of breaking the connection has a similar sequence of actions to when it was established.

## 5.3 DESPERANTO MEASUREMENT FACILITY DESIGN

In Chapter 4 an overview of the design of a distributed measurement facility for systems in the NAHCS class was given. The overview provided a summary of requirements for the PITL and end-users and described the general design of the measurement facility.

This section provides additional details on the design and operation of the measurement facility in the NAHCS class, using Desperanto to illustrate the design. The overall operation and the user interface design are discussed first. Next the measurement servers which provide the measurement and instrumentation functions are discussed followed by brief discussions of the control and the analysis functions. Detailed specification of the algorithms and data structures for the design are included in the appendices.

### 5.3.1 Measurement Facility Operation

The measurement facility has been designed as a comprehensive facility which meets the needs of the identified users. For the PITL and end-users a separate friendly interface must be provided which facilitates the use of the facility for that particular user. The facility does not require special training for the users.

#### 5.3.1.1 End-User Interface

The end-user desires measurement information on the response time for resources of interest. Since the end-user cannot be assumed to have a background in measurement, the measurement facility to user interface must be easy to use with maximum assistance available. The user only needs to be aware that measurement support does exist and knows how to initiate a request for measurement support.

The normal end-user will use a menu driven system to assist in constructing the request. For the end-user experienced in using the measurement capability the option of bypassing the detailed menu and issuing a single command to alleviate unnecessary prompting and iterative selection of measurement options is provided.

Figure 5.3 provides the measurement options and the format for measurement commands for starting and stopping measurements and for analyzing the collected data.

| OPTION NAME | OPTION VALUES | REMARKS |
|---|---|---|
| TYPE | Session* | Data entire session |
| | Request:id | Data for a single service (id) |
| ENDOPT | End | Stop at end of session |
| | Time:y | Stop in y minutes |
| | Count:x* | Stop after x requests |
| | Halt | Stop immediately |
| | Clock:z | Stop at clock time z |
| ANLYOPT | End* | Do when measurement stops |
| | Defer | Do upon user request |
| | Delete | Do not analyze data |
| LEVEL | All* | Display all parameters |
| | Select:x | Display parameters x |
| OUTPUT | Crt* | Display on Crt |
| | Print | Display on hard copy device |

* - default values

COMMAND FORMATS

```
START_USER_MEAS(TYPE,ENDOPT,ANLYOPT)
STOP_USER_MEAS(ENDOPT,ANLYOPT)
ANALYZE_USER_MEAS(LEVEL,OUTPUT)
```

Figure 5.3 Command Options and Formats for End-User Measurements

5.3.1.2 PITL Interface

In a similar fashion as for the end-user, the PITL requires an interface to the measurement facility that facilitates the use of it. Again a menu-driven interface with the option to override it for the experienced user is desired. To start a measurement session, the PITL must be logged into Desperanto but does not need to remain logged in during the duration of the session. The session can end at that point in time determined by the ENDOPT parameter in the measurement request. At a later time the PITL can again log into Desperanto and obtain analysis results from previous measurement session(s).

Figure 5.4 gives the options and formats for the PITL measurement commands.

| OPTION | OPTION VALUE | REMARKS |
|---|---|---|
| RESOURCE | xyz | Resource id |
| TYPE | Local* | Only owned resource |
| | Trace | Partial trace plus local |
| STARTOPT | Next* | Start at next request |
| | Count:x | Start after x requests |
| | Time:y | Start at system clock time y |
| INTERVAL | Count:x* | Store summary record |
| | Time:y | each x requests or y minutes |
| ENDOPT | Count:x* | Stop after x records |
| | Clock:z | Stop at clock time z |
| | Halt | Stop immediately |
| ANLYOPT | End | Do when measurement stops |
| | Defer* | Do upon user request |
| | Delete | Do not analyze data |
| LEVEL | All* | Display all parameters |
| | Select:x | Display parameters x |
| OUTPUT | Crt* | Display on Crt |
| | Print | Display on hard copy device |

* - default values

COMMAND FORMATS

```
START_PITL_MEAS(RESOURCE,TYPE,STARTOPT,INTERVAL,
                ENDOPT,ANLYOPT)
STOP_PITL_MEAS(RESOURCE,ENDOPT,ANLYOPT)
ANALYZE_PITL_MEAS(LEVEL,OUTPUT)
```

Figure 5.4 Command Options and Formats for PITL Measurement

### 5.3.2 Measurement Server

The measurement servers for the measurement facility are the processes which provide the measurement and most of the instrumentation functions. The servers are logically placed so that they are in the path of the distributed interactions for the system. The actual details of the implementation of the servers are dependent on the host system on which they are being placed. For systems that support such a capability, they will be processes which are created in response to a user request for measurement support and will remain active until the measurements are completed.

#### 5.3.2.1 End-User Measurement Server

The server for the end-user is placed in a logical position so that all requests for distributed support are routed through this server before they are acted on by the distributed system (see Figure 5.5). The measurement server port becomes the pseudo address for the distributed system server, the message is routed to this port, the appropriate data gathered, and then the message is forwarded on to the distributed server unchanged. When a response is returned for the end-user the port or address for the message known to the standard distributed server is again that of the measurement server. All communication between these two will be routed through the measurement server and the meas-

ures of interest will be collected at this point. Depending on the measurement parameter options selected by the end-user this server could remain in place throughout the distributed session or could be active for some portion of the session.

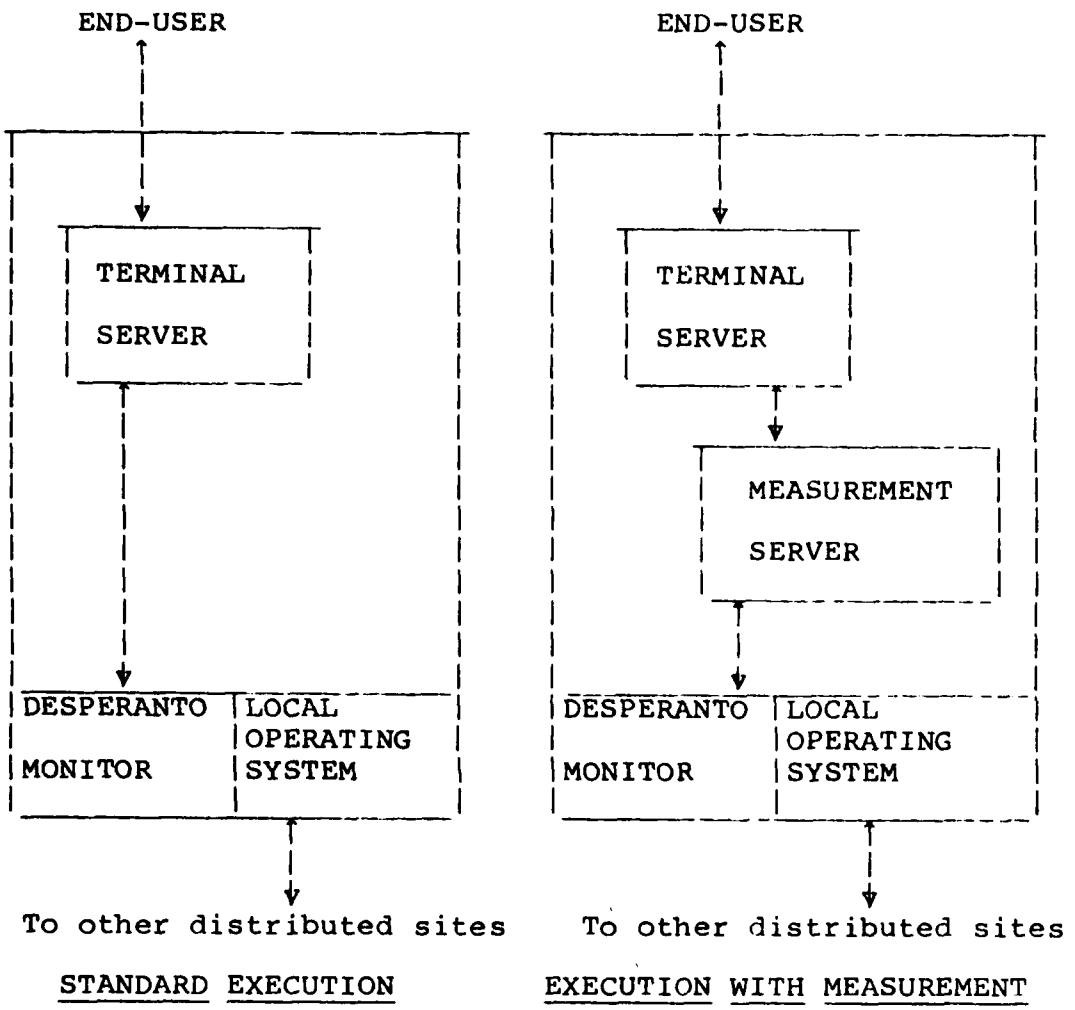



Figure 5.5 Measurement Server for the End-User

#### 5.3.2.2 PITL Server

For the PITL, the measurement server is placed between the standard resource interface server and the standard Desperanto monitor so that all requests for the resource are routed through the measurement server prior to being received by the standard resource interface server (see Figure 5.6). All responses from the resource are directed to the same measurement server prior to being sent to the requester using the distributed software system. The measurement server becomes the pseudo address for the resource's interface server. The measurement server provides the measurement function. In the case of the option to obtain usage measurements on a resource the instrumentation function will also be taken care of by the same server. On the other hand, for trace of resource requests the measurement function is handled by the measurement server associated at the resource but the instrumentation function will be distributed across the system. Each of these cases is addressed separately.

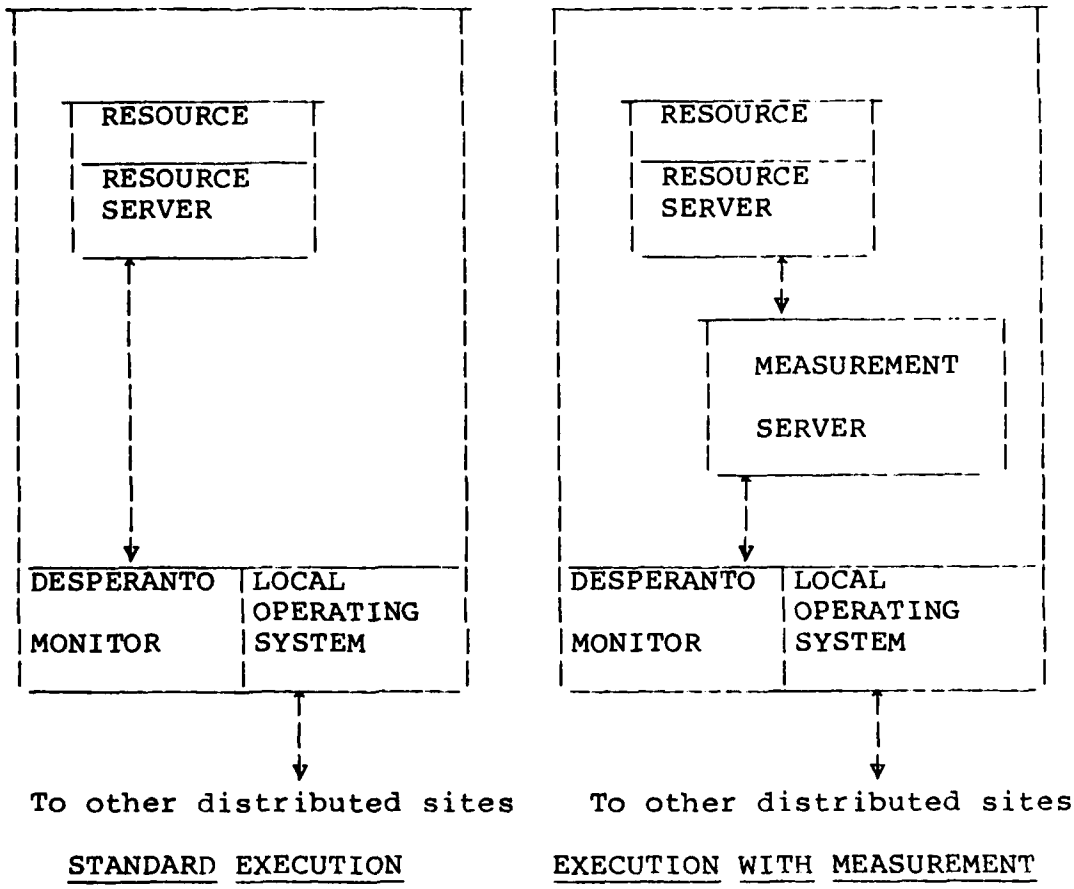



Figure 5.6 Measurement Server for the PITL

The option of gathering the data on resource usage involves creating a server which logically exists next to the server process for the resource so that all communication to and from the resource is routed through the measurement server. For long term measurements this means that each time the resource is invoked the measurement server is also invoked and placed in the appropriate location. An area of storage must be set aside for storing the data from the server and this data must be kept through many invocations of the resource. Hence some process which is active for an extended period must be used to facilitate this. The standard Desperanto monitor is that process. It has knowledge of the data storage and invokes the measurement server with pointers to the correct data area when a request for a resource is made.

The standard Desperanto monitor also is used to gather trace data. The measurement server placed between the resource and the Desperanto monitor will set a specific bit in the header of all messages requesting support for a remote resource to indicate that trace data is to be collected. Request messages which are received with the trace bit set will cause the standard Desperanto monitor at that site to save the system clock time. When the response message is returned to the requester the trace data is appended to the end of the response message by the standard monitor. Once

the response message is returned to the original requester the measurement data which has been appended to the end of the message will be removed by the measurement server and the appropriate measurement records updated.

### 5.3.3 Control

At each host the standard Desperanto monitor will maintain control over the measurements at that location. A request for measurement on a resource will be trapped by the standard Desperanto monitor and the necessary measurement servers invoked at the local host and at any remote hosts as required. Access rights are checked for each request and the entry for the resource in the Desperanto data base changed to indicate that the resource is being measured. Each standard Desperanto server maintains a list of all the resources at that location that are currently being measured and by whom the measurement request was made. When a request for the resource is made, the measurement server is invoked along with the resource server. Associated with this list will be information as to the location of measurement data associated with this resource including data which nas been moved to auxiliary storage.

### 5.3.4 Analysis

Analysis of measurement data collected by the measurement facility for systems in the NAHCS class does not differ from analysis of data from other systems. This is a well understood task and several good systems already exist that can be used to support this measurement function. No further design has been done.

## 5.4 MEASUREMENT PRIMITIVES

To aid the implementer of a measurement facility there needs to be specific basic tools from which the measurement facility can be built. These basic tools, measurement primitives, should be part of the system just as system service commands or hardware assists.

Kolanko [KOL77] in his research for the CNMS noted the need for formal specification techniques for the measurement and evaluation of computer networks. He identified a set of hardware and software modules necessary for network measurement (event detection, time measuring, counters, communication and control equipment, and other circuitry).

Recently Svoboda has also seen the need for measurement primitives and for measurement to be part of the overall system design [SVO81]. Her emphasis is on the use of a monitoring capability to support the measurement of reliability. A set of basic monitoring activities (event detection, event counting, direct timing, and sampling) were proposed as necessary for the task. To support these activities a basic set of resources (clocks, counters, accumulators, and mechanisms for event detection and signaling) are necessary. A method for providing this basic support was illustrated for a portion of a generic system.

Measurement primitives are the basic measurement resources that the individual programming the facility has available. Without defined primitives the programmer must construct the programs using the services provided by the host system and the programming language(s) chosen for implementation. If the programmer can abstract the task and use previously defined primitives then he/she can more easily implement the facility without worrying about, or even being aware of, the low level details of the program.

The primitives for the measurement facility consist of abstract data objects with operations defined on them. Formal work in defining abstract data types has been done by several researchers for languages such as CLU [LIS77] and Alphard [WUL76]. Ghezzi and Jazayeri [GHE82] discuss features of languages that support programming in the large. One feature that they identify as being desirable is information hiding - distinguishing between what a module exports for use by other modules and the internal details of how a module does the task.

For Desperanto the measurement requirements were analyzed and the measurement primitives needed were identified. Both the end-user measurement server and the PITL measurement server are specified in Appendix B using these primitives. At the level of abstract data objects this is a complete set of primitives.

The measurement primitives needed for the Desperanto system are listed in Figure 5.7. Each abstract data object(CLOCK, INTERVAL_TIMER, ACCUMULATOR) can have one or more occurrences in the measurement server. Each occurrence is referenced by a unique identifier. For each data object a set of valid operations are defined. Further definition is provided in Appendix A.

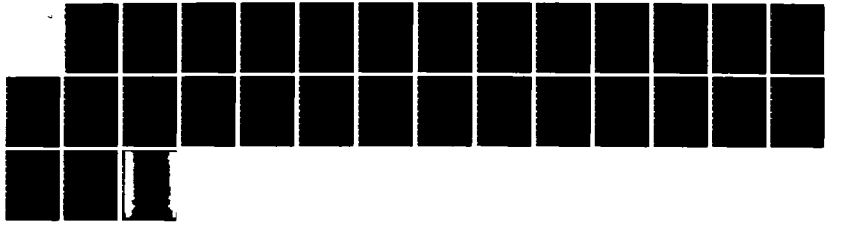
The objectives and implementations of the measurement facilities differ for the two referenced studies above from the Desperanto system. Yet, the primitives identified are comparable to those previously given in the literature. Time measurements and counting are common. Event detection is provided by the Desperanto monitor as is the case for many other systems.

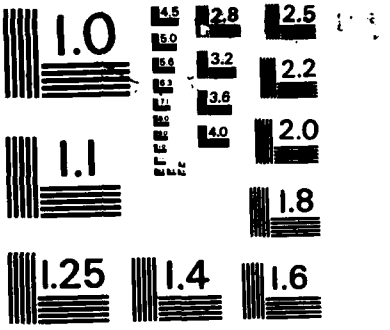
```
CLOCK

 CREATE_CLOCK(c)      -create clock c
 READ_CLOCK(c)        -read clock c, return time
 REMOVE_CLOCK(c)      -destroy clock c


INTERVAL_TIMER

 CREATE_IT(int)       -create interval timer int
 START_IT(int)       -start interval timer int
 READ_IT(int,val)     -read interval timer int,
                         return val
 REMOVE_IT(int)       -destroy interval timer int

ACCUMULATOR

 CREATE_ACC(acc)      -create accumulator acc
 RESET_ACC(acc)       -reset accumulator acc
 READ_ACC(acc,x:y)    -read accumulator acc,
                         return current value of
                         x:y(y=SUM,CNT,AVE,MIN,MAX,RGE)
 ADD_ACC(acc,z)       -add z to accumulator acc
 REMOVE_ACC(acc)      -destroy accumulator acc
```

Figure 5.7 Measurement Primitives

AD-A142 410 PERFORMANCE MEASUREMENT IN A DISTRIBUTED PROCESSING ENVIRONMENT(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH W E AYEN 1984 AFIT/CI/NR-84-10D 2/2

UNCLASSIFIED F/G 12/1 NL





## 5.5 SUMMARY

This chapter has described in detail the design for a specific example of a system in the NAHCS class of distributed environments. The design utilized the methodology approach presented in Chapter 4 along with the general design for systems in the NAHCS class from Chapter 4. A major content of the chapter was the identification of measurement primitives which can be used to easily construct measurement facilities as illustrated in Appendix B. The primitives can be programmed by the Desperanto implementers on those systems which are supporting the measurements. Once defined they can be used to implement the measurement facility per the design described earlier.

Appendix A provides further details on the measurement primitives. Appendix B gives detailed specifications for the control function and the measurement servers for the Desperanto measurement facility. These can be used during the implementation of measurement facilities.

# CHAPTER 6

# CONCLUSIONS AND FURTHER WORK

## 6.1 INTRODUCTION

This dissertation documents the results of research into the design of measurement facilities for a specific class of distributed computing systems. The class of systems is an important class that is attractive to many users but has limited popularity because of the difficulty in designing and implementing the software needed to interface the heterogeneous systems. The Desperanto distributed software support system is designed to remove these difficulties. The users of the software system will require tools to assist them in determining the performance of these systems. A measurement facility designed specifically for these systems will provide the system users with the ability to better manage the use of the system.

The research for this dissertation concentrated on defining the class of distributed systems and on identifying the measurement requirements for the measurement facility for the class. The facility design for systems in this class and the specific design for Desperanto was provided. This chapter highlights the contributions of the research and identifies further work.

## 6.2 RESEARCH CONTRIBUTIONS

NAHCS distributed systems includes those systems which are formed by connecting together existing computer sites by means of a communication subnetwork. The individual sites consist of heterogeneous architectures and are under managerial control of several organizations.

This is the first detailed investigation of measurement support for systems in this class. Previous researchers have addressed the measurement problem for systems with different characteristics and thus measurement requirements. The primary contributions are summarized below:

a. The dissertation provides the first description of the NAHCS class of distributed systems in terms of the characteristics which distinguish them from other systems. These distinguishing characteristics impose fundamental differences between systems in the class and those not in the class that are important from a measurement standpoint. A significant finding was that the granularity of the measurement is affected by system constraints, not by the measurement facility.

b. The users of the NAHCS measurement facilities were identified and their measurement requirements established. These users, the PITLs and end-users, differ in their measurement requirements both with respect to

each other and to measurement users outside the distributed computing system environment. The PITL was identified for the first time as a measurement user.

c. The design of measurement facilities for this class has been completed. After investigating all that is required by the users of the facility and considering the constraints, the resulting measurement requirements were simple and the resulting design was straight forward. This in contrast to what is normally expected of distributed systems and to what was expected when we began the research.

d. Measurement primitives were identified which are sufficient to implement the facility. These primitives are used to specify the measurement facility design for NAHCS distributed software support system.

e. The research contributes to the level of understanding with respect to the approach involved in designing measurement facilities for distributed systems. The measurement facility model presented in Chapter 4 provides a framework which can be used by designers of all measurement facilities, distributed or non-distributed. This simplifies the efforts required in facility design and can facilitate the sharing of knowledge about measurement facilities which will lead to a greater understanding of the measurement problem.

## 6.3 FURTHER WORK

This research has established the design for measurement facilities in the NAHCS class of distributed systems. During the development of the design a greater understanding of the characteristics of distributed systems was gained.

Further work is needed to investigate issues relative to the measurement facility. In particular, once the measurement facility is implemented then its performance can be evaluated using the tools available to the distributed system programmer. Additionally, the interface with, or in some cases support of, the accounting function for the distributed software system needs to be researched.

# APPENDIX A

## MEASUREMENT PRIMITIVES

This appendix provides a detailed description of the measurement primitives introduced in Chapter 5. The definition of a "primitive" and the use of them was described there and will not be repeated here. Appendix B illustrates the use of these primitives in the specification of measurement servers.

The following three sections give an overview of each primitive and in some cases example Pascal code to show a possible implementation of the primitive and to illustrate its structure. In an actual implementation the details will be hidden from the user and are system dependent. The data structures used are simple ones. More complex structures such as arrays, linked lists, and pointer vectors could be used if desired by a particular distributed system analyst.

## A-1. CLOCK

The CLOCK primitive should be provided to all measurement facility designers and implementors. Once created the instance of CLOCK can be used in other primitives or to obtain the current clock value for controlling measurement events. The CLOCK primitive converts the value and format of the system clock into a standard format for use in the measurement facility. The data structure for storing the value of the clock varies from system to system. The CLOCK primitive should remove differences in the interpretation of clock times due to specific implementations. For instance, the system clock can be defined to be the actual "wall clock" time, or the elapsed time since the system was last started, or the elapsed time since the clock was last read by that process.

On the DEC VAX the current value for the clock can be obtained by invoking the system service SYS$NUMTIM. This service returns the time in seven 16 bit words. Three of the words are used to keep the year, month, and day. The other four words contain the time in the format HOURS:MINUTES:SECONDS:HUNDREDTH OF A SECOND. The last four words can easily be converted to a single number containing the seconds (to one hundredth of a second) since midnight of that day.

The operations defined on the CLOCK primitive are as follows:

1. CREATE_CLOCK(c) - establish the data structure for the clock c. Data structure can be a single integer variable containing the time in hundredth of a second.

2. READ_CLOCK(c) - obtain the current value of the system clock, convert it into a standard form, and store it in the requisite data structure.

3. REMOVE_CLOCK(c) - destroy the data structure for clock c defined by the CREATE_CLOCK operation.

## A-2. INTERVAL_TIMER

The primitive INTERVAL_TIMER uses the CLOCK primitive to provide the capability of obtaining elapsed time values. This primitive should be used for determining end-to-end response times, checking for end of measurement interval, or for total measurement session time. A particular instance of the INTERVAL_TIMER can be reset without affect ; any other instances of it.

The operations defined on the INTERVAL_TIMER (IT, are as follows:

1. CREATE_IT(int) - create a data structure with the name 'int'. If a clock does not exist one must be created. The structure can be a single integer variable.

2. START_IT(int) - start the interval timer by storing the current value of the system clock obtained using READ_CLOCK in int.

3. READ_IT(int,val) - obtain the value for the interval int. The steps to take are as follows:

   - invoke READ_CLOCK

   - compare value stored in the interval timer with the clock value to see if midnight has passed

- calculate the time interval considering if midnight was passed.

- store the current time in int

- return the value for the interval to the user

A sample implementation for the READ_IT(int,val) is:

```
READ_IT (INT,VAL);
(* Read clock C and calculate the interval since *)
(* the last time saved in OLD_TIME               *)
(* Assume that the interval is always < 24 hours *)
(* MIDNIGHT contains the equivalent of 24 hours  *)
 BEGIN
   OLD_TIME := INT;
   READ_CLOCK(C);
   IF C > OLD_TIME THEN
     VAL := C - OLD_TIME
   ELSE
     VAL := MIDNIGHT -OLD_TIME + C;
   INT := C
 END
```

4. REMOVE_IT(int) - delete the references to it from the data structure. Invoke REMOVE_CLOCK if the clock was dedicated to this primitive.

## A-3. ACCUMULATOR

Many of the measurement parameters require maintaining counts of events or accumulating the sum of values and then determining the absolute value plus average, minimum,and maximum values for the parameters. The ACCUMULATOR primitive should provide the ability to obtain these parameters with little more effort than required for a normal assignment operation. For a defined instance of the data object the ADD operation causes the appropriate data within the data object to be changed. With the READ operation the current value for count(CNT), average(AVE), minimum(MIN), maximum(MAX), sum(SUM), and range(RGE) can be obtained. The RESET operation will restart the accumulation of the data.

The data structure for the ACCUMULATOR (ACC) could be a RECORD defined as:

```
ACCUM = RECORD
          COUNT,              (* count of entries        *)
          TOTAL,              (* sum of entries          *)
          MINIMUM,            (* minimum entry value     *)
          MAXIMUM: INTEGER (* maximum entry value     *)
        END;
```

The operations defined for this primitive are as follows:

1. CREATE_ACC(acc) - create the data structure with the name 'acc'. Initialize all elements in the structure.

2. RESET_ACC(acc) - reset all elements in acc to their

initial values.

3. ADD_ACC(acc,val) - update all elements in acc using the value val. The ADD_ACC operation can be implemented as follows:

```
ADD_ACC(ACC,ENTRY)
 (* assume that ACC is of type ACCUM and *)
 (* ENTRY is the value to be accumulated *)
 BEGIN
  WITH ACC DO
    COUNT := COUNT + 1;
    TOTAL : = TOTAL + ENTRY;
    IF ENTRY < MINIMUM THEN
      MINIMUM := ENTRY;
    IF ENTRY > MAXIMUM THEN
      MAXIMUM := ENTRY
  END
 END
```

4. READ_ACC(acc,fld:val) - access acc and return the value(s) stored in field(s) val. The second parameter can be implemented using vectors.

5. REMOVE_ACC(acc) - delete instance of acc.

# APPENDIX B

## DESPERANTO MEASUREMENT FACILITY

This appendix contains specifications for the Desperanto Measurement facility. The specification describes the operation of the control function located in the Desperanto monitor and the design of the measurement servers for the PITLs and the end-users. The description of the facility in Chapter 5 is assumed as a basis for the appendix. Also it is assumed that Desperanto has provided user interface servers, that the primitives introduced in Chapter 5 and Appendix A are available, and that the analysis is accomplished using existing analysis packages.

The organization of the appendix is as follows:

B-1 CONTROL

The control function is included in the Desperanto monitor. At the time that a measurement user requests a measurement with either the START_USER_MEAS or START_TOOL_MEAS commands the request is trapped and the appropriate module is invoked. The same action takes place when any of the other measurement commands are issued. For all commands, the measurement user, PITL or end-user, is prompted using the menu displays for parameters which are missing or are incorrect.

The control module will perform the following functions relative to the measurement servers.

1. Check the request for correct entries and request any additional entries or modifications necessary.

2. Invoke the appropriate measurement servers and set the parameters in the Desperanto system to handle the measurements.

   a. Entries in the Desperanto data base are changed to indicate that resource is being measured and required storage is allocated.

   b. For end-user measurements the measurement server is invoked and it's address is the pseudo address for the user's terminal server.

c. For the PITL measurements the measurement server is invoked and it's address is the pseudo address for the resource server.

4. During measurement activity the measurement servers detect the events of interest, collect the data, and write the data to storage. The control function can be used to assist the servers in controlling the length of measurement sessions and in writing data records to storage.

5. At the completion of the measurement session:

a. A message is sent to the active measurement server to cause a final measurement record to be written..

b. The Desperanto data base is modified as required.

c. Measurement servers are deactivated.

## B-2 DESPERANTO MEASUREMENT SERVER

At the time that a request for measurement on a resource is made a measurement server is established as discussed in Chapter 5 and B-1. These servers will exist, although not always active, until the measurement terminates. The address for the resource or the end-user's process is the address of the measurement server.

The specification for the design of the servers is divided into three parts - start of measurement session, the session itself , and end of session. Features unique to just one server, PITL or end-user, are noted.

1. Initialization of measurement session

-commands used - START_PITL_MEAS or START_USER_MEAS

- verify that resource xyz is a valid resource for measurement (PITL only)

- establish start conditions (STARTOPT)

```
If option = next then
  Initiate measurement immediately

If option = count then
  CREATE_ACC(start_cnt)
  If val from READ_ACC(start_cnt,val:CNT) < x
    invoke ADD_ACC(start_cnt)
    for each request received
  Start measurement
```

```
    If option = time then
      CREATE_CLOCK(start_clk)
      While start_clk < y wait
      Start measurement
```

- Create the necessary data structures

```
    CREATE_CLOCK(xyz_resp)   (*response time    *)

    CREATE_ACC(xyz_usage)    (*request count    *)

    CREATE_ACC(xyz_wait)     (*wait time - PITL*)

    CREATE_ACC(xyz_comm)     (*communication time *)
                             (* - PITL only        *)

    CREATE_ACC(user_resp)    (*user response time *)
```

- Establish end of measurement option

```
    If option = count then
      CREATE_ACC(stop_cnt)
      If val from READ_ACC(stop_cnt,val:CNT) < x
        then ADD_ACC(stop_cnt,1)
        else stop measurement

    If option = clock then
      Do same as for STARTOPT

    If option = halt then
      Stop immediately
```

-Establish mechanism for measurement record preparation

```
    If option = count then
      CREATE_ACC(rec_cnt)

    If option = time then
      CREATE_ACC(rec_clk)
```

2. During a measurement session each activity relative to the resource requires measurement server response.

- use primitives to update all data structures initialized during start of session (xyz_usage,xyz_resp, etc.)

-If trace option invoked (PITL only) then

If action is request for service from another resource then
set trace bit in header of request message

If action is response to request then
process all data structures using the primitives.

-Check to determine if data record should be written

If INTERVAL option = count and rec_cnt - x > 0 then
Write record

If INTERVAL option = time and rec_clk - y > 0 then
Write record

3. At end of measurement session

Write final measurement record

Invoke analysis service if ANLYOPT = end

Delete all measurement records if ANLYOPT = delete

BIBLIOGRAPHY

[ABR77] Abrams, M.D., Cotton, I.W., Watkins, S.W., Rosenthal, R., and Rippy, D.E., "The NBS Network Measurement System," IEEE Transactions on Computers, Vol COM-25, No. 10, October 1977, pp.1189-1198.

[ABR79] Abrams, Marshall D., and Neiman, Dorothy C., "The NBS Network Measurement Instrument," Proceedings of the 15th CPEUG Meeting," San Diego, California, October 15-18, 1979, pp.201-211. (Also published as NBS Pub 500-52, October 1979)

[AME82] Amer, Paul D., "A Measurement Center for the NBS Local Area Computer Network", IEEE Transactions on Computers, Vol. C-31, No. 8, August 1982, pp. 723-729.

[BAR68] Barnes, G.H., Brown, R.M., Kato, D.J., Slotnick, D.L., and Stokes, R.A., "The ILLIAC IV Computer", IEEE Transactions on Computers, Vol. C-17, August 1968, 746-757.

[BLA80] Blake, Russ, "XRAY: Instrumentation for Multiple Computers," Proceedings of Performance '80, Toronto, Ontario, Canada, May 28-30, 1980, pp. 11-25.

[BOO81] Booth, Grayce, M. The Distributed System Environment: Some Practical Approaches, McGraw-Hill, New York, 1981.

[BOR79] Borovits, Israel and Neumann, Serv, Computer Systems Performance Evaluation, Lexington Books, Massachusetts, 1979.

[BRO76] Browne, J.C., "A Critical Overview of Computer Erformance Evaluation," 2nd International Conference on Software Engineering, October 13-15, 1976, pp. 138-145.

[BUC77] Buck, David L., and Hrynyk, David M., "Software Architecture for a Computer Network Monitoring System," International Conference on Performance of Computer Installations, Ferrari, D., editor, 1978, pp. 269-287.

[BUZ76] Buzen, J. P., "Fundamental Laws of Computer System Performance," Proceedings of the International Symposium on Computer Performance Modeling, Measurement, and Evaluation, Chen, Peter P. S., and Franklin, Mark, editors, Boston, Massachusetts, March 29-31, 1976, pp. 200-210.

[CHE79] Cheriton, D. R., Malcolm, M. A., Melan, L. S., and Sager, G. R., "Thoth, A Portable Real-Time Operating System," Communications of ACM, Vol. 22, No. 2, February 1979, pp. 105-114.

[COL71] Cole, Gerald D., "Computer Network Measurement Techniques and Experiments," UCLA- ENG-7165, October 1971.

[COT77] Cotton, Ira. W., "Local Area Networking : Report of a Workshop Held at the National Bureau of Standards," Gaithersburg, Maryland, August 22-23, 1977. (NBS Special Publication 500-31)

[DER76] DeRemer, Frank, and Kron, Hans. H., "Programming-in-the-Large Versus Programming-in-the-Small," IEEE Transactions on Software Engineering, Vol. SE-2, No. 2, June 1976, pp. 80-86.

[DES80] desJardins, Richard, and White, George W., "ISO/ANSI Reference Model of Open Systems Architecture," Proceedings of Trends and Applications:1980 National Bureau of Standards, May 29, 1980, pp. 47-53.

[DRU73] Drummond, M. E. Jr., Evaluation and Measurement Techniques for Digital Computer Systems, Prentice-Hall, New Jersey, 1973.

[ECK78] Eckhouse, Richard H. Jr. and Stankovic, John A., "Issues in Distributing Processing - An Overview of Two Workshops," IEEE Computer, January 1978, pp. 222-26.

[ENS78] Enslow, Phillip H. Jr., "What is a 'Distributed' Data Processing System?," Computer, January 1978, pp. 13-21.

[FER78] Ferrari, Domenico, Computer Systems Performance Evaluation, Prentice-Hall, New Jersey, 1978.

[FOR78] Forsdick, Harry C., Schantz, Richard E. and Thomas, Robert H., "Operating Systems for Computer Networks," IEEE Computer, January 1978, pp. 48-57.

[GEH82] Gehringer, Edward F., Jones, Anita K., and Segall, Zary Z., "The Cm* Testbed," IEEE Computer, Vol. 15, No. 10, October 1982, pp. 40-53.

[GHE82] Ghezzi, Carlo, and Jazayeri, Mehdi, Programming Language Concepts, John Wiley, New York 1982.

[HRY78] Hrynyk, David M., "A Computer Network Measurement Definition and Control Language and Compiler," CCNG Report T-25, University of Waterloo, January 1978.

[IBM77] IBM OS/VS2 MVS Resource Management Facility (RMF) Version 2 General Information Manual, GC28-0921-0.

[JON80] Jones, Anita K. and Schwarz, Peter, "Experience Using Multiprocessor Systems - A Status Report," Computing Surveys, Vol. 12, No. 2, June 1980, pp. 121-165.

[KAT78] Katzman, James A., "A Fault Tolerant Computing System," IEEE Hawaii International Conference of System Sciences, January 1978.

[KIM78] Kimbleton, Stephen R., Wood, Helen M., and Fitzgerald, M. L., "Network Operating Systems - An Implementation Approach," AFIPS National Computer Conference Proceedings, 47, June 1978, pp. 773-782.

[KOL77] Kolanko, Richard, "A Structured Approach to Performance Measurement of Computer Networks," CCNG Report T-62, University of Waterloo, June 1977.

[LAN82] Lantz, Keith A., Gradischnig, Klaus D., Feldman, Jerome A., and Rashid, Richard F., "Rochester's Intelligent Gateway," IEEE Computer, Vol. 15, No. 10, October 1982, pp. 54-68.

[LAZ81] Lazowska, Edward D., Levy, Henry M., Almes, Guy T., Fischer, Michael J., Fowler, Robert J., and Vestal, Stephan C., "The Architecture of the Eden System," Proceedings of the Eighth Symposium on Operating Systems Principles, Asilomar, December 1981, pp. 148-159.

[LIS77] Liskov, B., Snyder, A., Atkinson, R., and Schaffert, C., "Abstraction Mechanism in CLU," CACM, Vol. 20, No. 8, August 1977, pp. 564-576.

[MCD77] McDaniel, Gene, "METRIC: A Kernel Instrumentation System for Distributed Environments," Proceedings of the Sixth ACM Symposium on Operating System Principles, November 1977, pp. 93-99.

[MAM81] Mamrak, S. A., "Installing Existing Tools in a Distributed Processing Environment," Proceedings of the First Conference on Foundations of Software Technology and Theoretical Computer Science, Bangalore, India, December 1981, pp. 49-56.

[MAM82a] Mamrak, Sandra A., Ayen, W. E., Gherfal, F., and Leinbaugh, D., "Performance Measurement and Exception Handling in Desperanto's Distributed Environment," Proceedings 3rd International Conference on Distributed Computer Systems, Miami, Florida, October 1982, pp. 840-846.

[MAM82b] Mamrak, S. A., Maurath,P., Gomez, J., Janaradan, S., and Nicholas, C., "Guest Layering Distributed Processing Support on Local Operating Systems," Proceedings 3rd International Conference on Distributed Computer Systems, Miami, Florida, October 1982, pp. 854-859.

[MAM82c] Mamrak, S. A., Kuo, J., and Soni, D., "Supporting Existing Tools in Distributed Processing Systems: The Conversion Problem," Proceedings of the 3rd International Conference on Distributed Computer Systems, Miami, October 1982, pp. 847-853.

[MAM83] Mamrak, Sandra A., Leinbaugh, Dennis, and Berk, Toby S., "A Progress Report on the Desperanto Research Project Software Support for Distributing Processing," ACM Operating Systems Review, Vol. 17, No. 1, January 1983, pp. 17-29.

[MOR75] Morgan, David E., Banks, Walter, Goodspeed, Dale P., and Kolanko, Richard, "A Computer Network Monitoring System," IEEE Transactions on Software Engineering, Vol.SE-1, No. 3, September 1975, pp. 299-311.

[NUT75] Nutt, Gary J., "Tutorial: Computer System Monitors," IEEE Computer, November 1975, pp. 51-61.

[OUS80] Ousterhout, Jack K., Scelza, Donald A., and Sindhu, Pradeep S., "Medusa: An Experiment in Distributed Operating System Structure," Communications of the ACM, Vol. 23, No. 2, February 1980, pp. 92-105.

[POP81] Popek, G., Walker, B., Chow, J. Edwards, P., Kline, C., Rudisin, G., and Thiel, G., "LOCUS: A Network Transparent, High Reliability Distributed System," Proceedings Eighth Symposium on Operating Systems Principles, Asilomar, December 1981, pp. 169-177.

[RAS78] Raskin, Levy, "Performance Evaluation of Multiple Processor Systems, PhD Thesis, Carnegie-Mellon U., August 1978.

[ROS78] Rose, C. A., "A Measurement Procedure for Queueing Network Models of Computer Systems," ACM Computing Surveys, Vol. 10, No. 3, September 1978, pp. 263-280.

[SCH80] Schantz, R., and Swernofsky, S., "The TENEX/TOPS-20 NSW Foreman Program Maintenance Manual: Appendix A Performance Measurement Package," BBN Report No. 4093, May 1980.

[SHO80] Shoch, J. F. and Hupp, J. A., "Measured Performance of an Ethernet Local Network," ACM Communications, Vol. 23, No. 12, December 1980, pp.711.

[SVO76] Svobodova, Liba, Computer Performance Measurement and Evaluation Methods: Analysis and Applications, Elsevier, New York, 1976.

[SVO80] Svobodova, Liba, "Performance Problems in Distributed Systems," INFOR, Vol. 18, No. 1, February 1980.

[SVO81] Svobodova, Liba, "Performance Monitoring in Computer Systems - A Structured Approach," ACM Operating Systems Review, Vol. 15, No. 3, July 1981, pp. 39-50.

[SWA77] Swan, R.j., Bechtholsherim, A., Lai, K.W., and Ousterhout, J.K., "The Implementation of the Cm* Multiprocessor", Proceedings AFIPS 1977 National Computer Conference, Vol. 46, AFIPS Press, Montvale, N.J., 1977, 645-655.

[TAN81] Tanenbaum, Andrew S., Computer Networks, Prentice-Hall, New Jersey, 1981.

[THO73] Thomas, Robert H., "A Resource Sharing Executive for the ARPANET," AFIPS National Computer Conference Proceedings 42, 1973, pp. 155-163.

[TRI78] Tripathi, Anand P., Upchurch, Edwin T., and Browne, James C., "An Overview of Research Directions in Distributing Processing," IEEE COMPCON Fall 78.

[UNI78] "UNIVAC Software Instrumentation Package," Univac Corporation, March 1978.

[WAL80] Wallack, Barry M. and Gero, George H., "Generalized Monitoring Facility Users Manual," DOD Command and Control Technical Center Manual CSM UM 246-80, July 1980.

[WAT72] Watson, W.J., "The TI ASC - A Highly Modular and Flexible Supercomputer Architecture", Proceedings AFIPS 1972 Fall Joint Computer Conference, Vol. 41, AFIPS Press, Montvale, N.J., 1972, 221-228.

[WAT80] Watson, Richard W. and Fletcher, John G., "An Architecture for Support of Network Operating System Services," Computer Networks, Vol. 4, No. 1, February 1980, pp. 33-49.

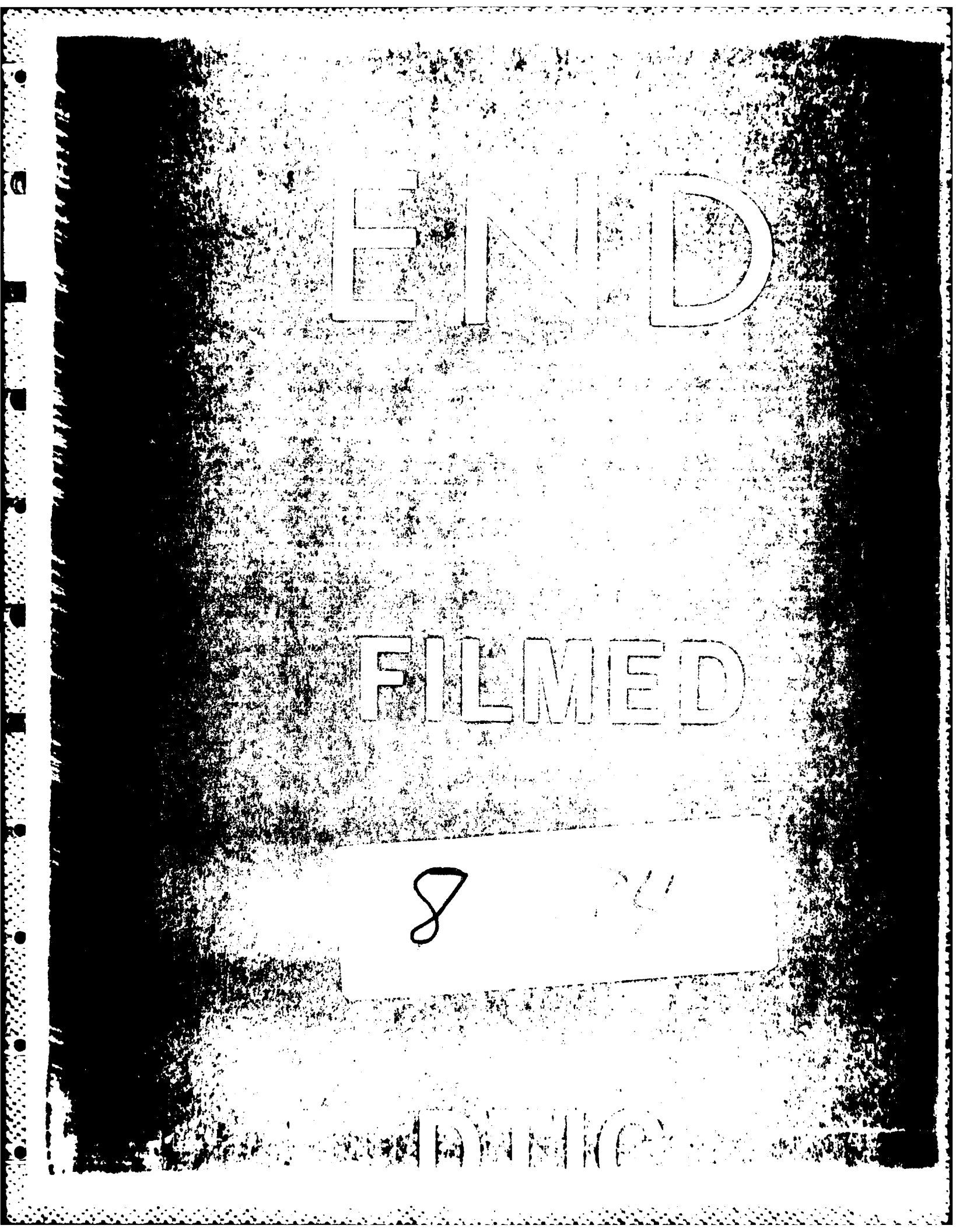
[WEI80] Weitzmann, Cay, Distributed Micro/Minicomputer Systems: Structure, Implementation, and Applications, Prentice-Hall, New Jersey, 1980.

[WUL72] Wulf, W.A., and Bell, C.G., "C.mmp - A Multimini Processor", Proceedings AFIPS 1972 Fall joint Computer Conference, Vol. 41, AFIPS Press, Montvale, N.J., 1972, 765-777.

[WUL77] Wulf, W. A., Linden, R., and Shaw, M., "An Introduction to the Construction and Verification of Alphard Programs," Communications of ACM, Vol. 20, No. 8, August 1977, pp. 253-264.

[WUL81] Wulf, William A., Levin, Roy, and Harbison, Samuel P., HYDRA/C.mmp: An Experimental Computer System, McGraw-Hill, New York, 1981.

[ZIM80] Zimmerman, H., "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection," IEEE Transactions on Communications, Vol. COM-28, No. 4, April 1980, pp. 425-432.


END
FILMED
8